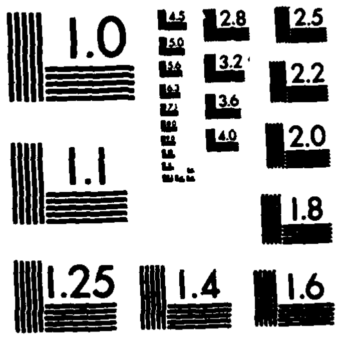
AD-A123 818 PROJECTION OF MAXIMUM SOFTWARE MAINTENANCE MANNING LEVELS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S L TODY ET AL. JUN 82 1/2

UNCLASSIFIED F/G 9/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

(2)

AD A123818

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A

PROJECTION OF MAXIMUM SOFTWARE MAINTENANCE MANNING LEVELS

by

Stephen L. Tody

Ray A. Hodgson

June 1982

Thesis Advisors: Ron Modes
Dan C. Boger

83 01 26 069

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD A123818	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PROJECTION OF MAXIMUM SOFTWARE MAINTENANCE MANNING LEVELS		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Stephen L. Tody Ray A. Hodgson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1982
		13. NUMBER OF PAGES 145
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Maintenance
Software Life Cycle
Manpower Estimation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The apparent lack of management of software maintenance within DOD and throughout the software industry has given rise to concern, as the costs associated with software maintenance continue to increase. The major contributor to the rise in maintenance costs seems to be personnel costs as opposed to hardware acquisition or computer time. However, to-date, it appears that little research has been conducted to attempt to resolve this problem. There also appears to be a lack of any standard definition

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. (continued)

of software maintenance. This thesis discusses various models which have been developed to attempt to predict maintenance manloading as the controlling factor in maintenance costing. It evaluates one model in particular, and proposes a possible maintenance versus life cycle phase relationship which may be of assistance to the software manager in maintenance manloading prediction. It also proposes specific topics for further research in this area.

A

Projection of Maximum Software Maintenance Manning Levels

by

Stephen L. Tody
Lieutenant Commander, United States Navy
B.S., Trinity College, 1970

and

Ray A. Hodgson
Captain, United States Army
B.S.I.T., Kansas State College, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
June 1982

Authors: Stephen L. Tody

Ray A. Hodgson

Approved by: Dan C. Boger
Co-Thesis Advisor

[illegible]
Co-Thesis Advisor

[illegible]
Chai~~rman~~, Department of Administrative Sciences

[illegible]
Dean of Information and Policy Sciences

ABSTRACT

The apparent lack of management of software maintenance within DOD and throughout the software industry has given rise to concern, as the costs associated with software maintenance continue to increase. The major contributor to the rise in maintenance costs seems to be personnel costs as opposed to hardware aquisition or computer time. However, to-date, it appears that little research has been conducted to attempt to resolve this problem. There also appears to be a lack of any standard definition of software maintenance. This thesis discusses various models which have been developed to attempt to predict maintenance manloading as the controlling factor in maintenance costing. It evaluates one model in particular, and proposes a possible maintenance versus life cycle phase relationship which may be of assistance to the software manager in maintenance manloading prediction. It also proposes specific topics for further research in this area.

TABLE OF CONTENTS

I. INTRODUCTION 9
A. BACKGROUND 9
B. PROBLEM DEFINITION 16
C. RESEARCH OBJECTIVES 17
D. ASSUMPTIONS/LIMITATIONS 18
E. RESEARCH METHODOLOGY 19
F. OVERVIEW OF THE THESIS 22
II. SOFTWARE MAINTENANCE COST ESTIMATION MODELS . . . 23
A. CURRENT TECHNIQUES USED AS A BASIS FOR THE GREEN/SELBY MODEL 23
1. Putnam's Software Cost Estimating Model . 23
2. Parr's Software Cost Estimating Model . . 26
3. Army Macro-estimating Model 29
4. The Lehman-Belady Model 31
B. OTHER MODELS OF INTEREST 35
1. Jensen Model 35
2. Other Models 44
C. CHAPTER TWO SUMMARY 44
III. MAINTENANCE COST ESTIMATING VIA THE GREEN/SELBY MODEL 47
A. MACRO APPROACH 47
B. MICRO APPROACH 50
C. PROJECTED MODEL APPLICATIONS 51
D. CHAPTER THREE SUMMARY 52
IV. MODEL VALIDATION 53
A. MATHEMATICAL DEVELOPMENT 53
B. EMPIRICAL DEVELOPMENT 60
C. CHAPTER FOUR SUMMARY 62
V. RESEARCH ANALYSIS 65
A. DATA DEFINITION 65
1. Bankers Trust Co. Data 65

2. NASA data 65

B. ANALYSIS PROCESS 66

C. ANALYSIS RESULTS 68

D. CHAPTER FIVE SUMMARY 72

VI. CONCLUSIONS AND RECOMMENDATIONS 73

A. INTRODUCTION 73

B. CONCLUSIONS 73

C. RECOMMENDATIONS 77

APPENDIX A: ANALYSIS OF SOFTWARE MODELS BY THIBODEAU . . 80

A. INTRODUCTION 80

B. AEROSPACE MODEL 80

C. DOD MICRO ESTIMATING PROCEDURE 81

D. DOTY ASSOCIATES, INC. 90

E. FARR AND ZAGORSKI MODEL 100

F. WOLVERTON 101

APPENDIX B: ANALYSIS OF SOFTWARE MODELS BY WOLVERTON . .106

A. INTRODUCTION .106

B. BOEING COMPUTER SERVICE COST MODEL 106

C. IBM WALSTON-FELIX COST MODEL 111

D. PUTNAM'S SOFTWARE LIFE CYCLE COST MODEL (SLIM) 116

APPENDIX C: SUPPORTING DATA AND CURVES 128

LIST OF REFERENCES .141

INITIAL DISTRIBUTION LIST 144

LIST OF TABLES

I. Cost Models 11
II. Ordinates for Manpower Functions 30
III. Project Complexity Values 38
IV. Technology Constant Adjustment Factors 41
V. Bankers Trust Co. Projects Characteristics 66
VI. NASA Project Data Characteristics 67
VII. Compilation of Analysis Results 71
VIII. Labor Estimate Adjustment Factor 109
IX. Forecasted versus Actual Costs for the BCS Model 110
X. Project A Data 128
XI. Project B Data 129
XII. Project C Data 130
XIII. Project D Data 131
XIV. Combined Project A-D Data Normalized to td=1 . . 132
XV. NASA Project Data 134
XVI. Project A Curves 135
XVII. Project B Curves 136
XVIII. Project C Curves 137
XIX. Project D Curves 138
XX. Combined Project A-D Curves 139
XXI. Nasa Project Curves 140

LIST OF FIGURES

2.1 Rayleigh Project Life Cycle Profile 23

2.2 Comparison of Sech^2 and Rayleigh Curves 28

2.3 Macro-Estimating Model Solution Surface (Graphical Representation) 42

2.4 Feasible Solution Region 43

2.5 Risk Analysis of Schedule Using Graphical Technique 45

3.1 Normalized Rayleigh Curve 50

4.1 Maintenance Phase Timing Relationships 56

4.2 Maintenance Phase Timing Relationships in the Green/Selby Model 57

4.3 Developed Normalized Curve 63

5.1 NASA Data Fitted Curves 69

I. INTRODUCTION

A. BACKGROUND

The department of defense for the last twenty to thirty years has become more and more reliant on automatic data processing equipment to accomplish its seemingly ever increasing and complex mission. When this trend started, hardware was the overriding concern, consuming, in 1955, more than 80 percent of the data processing dollar [1]. Through the years, technical inovations, such as the evolution from vacumm tubes to discrete transistors and from discrete transistors to integrated circuits, coupled with the increased use of mass production have decreased the cost of hardware. However, software has continued to rise in price. This rise in the price of software and the decrease in the price of hardware has resulted in software rapidly becoming the more costly of the two, and it is predicted that by 1985 it will account for better than 90 percent of the data processing dollar [2].

The true impact of this development may not appear to be significant until one realizes that the value of this software in 1973 was set at 20 billion dollars for the United States [3], and is estimated to be over 200 billion dollars in 1985 [4].

As a direct result of the monetary value of software production, many techniques have been developed to estimate, at the start, what the overall life cycle cost of a software project will be. A recent study conducted by Hughes Aircraft Company for the Air Force examined twenty-one of these models to determine commonalities and differences in their cost estimating approaches. Ten of these models are limited to software development cost, while eleven have software support cost as a primary or secondary output. Table I lists all of the models studied, in alphabetical order.[5]

Originally, it was thought that development costs were the most important item to derive and/or estimate. In fact, the development and design efforts for a new system are indeed still looked upon as more enjoyable and rewarding than the maintenance effort for an existing system. There are, of course, many reasons for this view. Six of these reasons, according to Robert Glass, are :

1. Maintenance is intellectually very difficult. Problems cannot be bounded. The cause could be anywhere.
2. Maintenance is technically very difficult. Problems cannot be specialized. They could surface because of errors in the coding, design, architecture, or concept.
3. Maintenance is unfair. Usually the person who is maintaining a product did not write it and must interpret what the original author meant. Documentation is inadequate most of the time.
4. Maintenance is no - win. People only come to maintenance with problems.
5. Maintenance is infamous. There is very little glory, noticeable progress, or chance for 'success'.
6. Maintenance lives in the past. The general quality of code being maintained is often terrible. This is partly because it was created when everybody's understanding of software was more rudimentary, and partly because a great deal of code is produced by people before they become really good at programming.[6]

TABLE 1

COST MODELS

MODEL NOMENCLATURE & USER	APPLICATION	TECHNIQUES	DATA REQUIRED	COMMENTS
AEROSPACE MODEL, AIR FORCE AVIONICS LABORATORY/AEROSPACE CORPORATION	MANMONTH ESTIMATES FOR DESIGN, DEVELOPMENT, OPERATION AND MAINTENANCE	TOP-DOWN BASED ON NUMBER OF INSTRUCTIONS	NUMBER OF INSTRUCTIONS	COST EQUATIONS FOR REAL-TIME AND SUPPORT PROGRAMS; INCLUDES GROSS SOFTWARE MAINTENANCE CALCULATIONS
SOFTWARE DEVELOPMENT COST METHOD BOEING	MANHOUR ESTIMATES FOR DEVELOPMENT OUTPUTS	TOP-DOWN BASED ON NUMBER OF INPUTS AND	NUMBER OF INPUTS/OUTPUTS	SINGLE EQUATION: MH= .33I**2, WHERE I= INPUTS + OUTPUTS
SOFTWARE MILITARY SUPPORT COST METHOD, BOEING	SUPPORT PERSONNEL REQUIREMENTS ESTIMATION	TOP-DOWN BASED ON NUMBER OF INSTRUCTIONS	NUMBER OF INSTRUCTIONS	SINGLE EQUATION: NSP=I/1000 WHERE I= NUMBER OF INSTRUCTIONS
COMPUTER RESOURCES INTEGRATED SUPPORT PLAN MODEL (CRISP), BOEING	SOFTWARE LIFE CYCLE COSTING	BOTTOM-UP	NUMBER OF INSTRUCTIONS CHANGED, SHIFTS PER INSTRUCTION, MANHOURS PER SHIFT, MANHOURS PER MONTH, MANHOURS PER LINE OF CODE	NOT SUITABLE FOR PREDICTING COSTS BUT GOOD APPROACH FOR TRACKING COSTS
C-14 MODEL, BOEING	SUPPORT COSTS ESTIMATES	BOTTOM-UP	NUMBER OF INSTRUCTIONS CHANGED, MANMONTHS, AND COSTS PER MANMONTH PLUS OTHER SUPPORT COSTS	COST OF SUPPORT = (COST/MANMONTH X MANMONTHS REQUIRED) + ADDITIONAL SUPPORT COSTS
ESD MODEL, ELECTRONIC SYSTEM DIVISION	LISTING OF FACTORS AND HARDWARE DEVELOPMENT COST	TOP-DOWN BASED ON NUMBER OF INSTRUCTIONS	NUMBER OF INSTRUCTIONS LANGUAGE, APPLICATIONS, ETC.	SUBJECTIVE PARMETERS INPUT TO ALGORITHMS REFLECT DEVELOPMENT TECHNIQUE, PERSONNEL SKILLS, ETC.
CONCEPT MODEL ELECTRONIC SYSTEMS DIVISION	LIFE CYCLE COST ESTIMATION	TOP-DOWN BASED ON NUMBER OF INSTRUCTIONS	NUMBER OF INSTRUCTIONS	DEVELOPED FOR ESD BY DOTY ASSOCIATES; GOOD APPROACH FOR SOFTWARE LCC ESTIMATION
F-16 MODEL, AIR FORCE AVIONICS LABORATORY	MANMONTH ESTIMATES FOR OPERATION AND SUPPORT	TOP-DOWN BASED ON NUMBER INSTRUCTIONS	NUMBER OF INSTRUCTIONS	SINGLE EQUATION: MN= .00712 X I WHERE I= NUMBER OF INSTRUCTIONS
GRC MODEL, GENERAL RESEARCH CORPORATION	MANMONTH ESTIMATES FOR DEVELOPMENT PROGRAMS	TOP-DOWN BASED ON NUMBER OF INSTRUCTION AND	NUMBER OF INSTRUCTIONS, LANGUAGE, DEGREE OF DIFFICULTY, DEGREE OF HOST COMPUTER SATURATION	SINGLE EQUATION: T= 0.04N ** 0.79 X 0.54 ** L WHERE N= NUMBER OF INSTRUCTIONS AND L= 1 FOR HIGH LEVEL LANGUAGE AND ZERO FOR ASSEMBLY LANGUAGE

TABLE 1 CONTINUED

MODEL NOMENCLATURE & USER	APPLICATION	TECHNIQUES	DATA REQUIRED	COMMENTS
GRC MODEL (VERSION A), GENERAL RESEARCH CORPORATION	RESOURCE CONSUMPTION ESTIMATES BY PROGRAM PHASE	TOP-DOWN BASED PRIMARILY ON NUMBER OF INSTRUCTIONS	NUMBER OF INSTRUCTIONS, LANGUAGE, DEGREE OF DIFFICULTY, DEGREE OF SATURATION OF HOST COMPUTER	ERROR CORRECTION IS CONSIDERED ONLY LEGITIMATE MAINTENANCE EFFORT; MODIFICATIONS ARE DEFINED AS DEVELOPMENT EFFORTS
GRC MODEL (VERSION B), GENERAL RESEARCH CORPORATION	ERROR CORRECTON COST ESTIMATION SUBSEQUENT TO DELIVERY	TOP-DOWN BASED ON NUMBER OF INSTRUCTIONS	NUMBER OF INSTRUCTIONS	SINGLE EQUATION: COST = 1740 X NUMBER OF INSTRUCTIONS X (-.0011) X LIFE CYCLE YEAR + 0.0161))
HUGHES MODEL, HUGHES AIRCRAFT COMPANY	RESEARCH, DESIGN, TEST, AND EVALUATION COST ESTIMATION	TOP-DOWN USING COST ESTIMATION RELATIONSHIPS	SOFTWARE, HOST COMPUTER AND DEVELOPMENT PROGRAM DESCRIPTIONS	EXTENSIVE HISTORICAL DATA USED FOR DEVELOPMENT
COST ESTIMATION MODEL, IBM	DEVELOPMENT COST ESTIMATION	TOP-DOWN USING NUMBER OF LINES	SOFTWARE DESCRIPTION	SINGLE EQUATION: MANMONTHS = 5.2 L**0.91 WHERE L = NUMBER OF LINES
COST OF FUNCTION MODEL, NAVAL AIR DEVELOPMENT CENTER	ESTIMATION OF TOTAL ACQUISITION COSTS: RESEARCH DEVELOPMENT, TEST AND EVALUATION PLUS PRODUCTION	BOTTOM-UP BASED ON SIX VARIABLES (SEE DATA REQUIREMENTS)	NUMBER OF INSTRUCTIONS, CONTRACTOR TRAVEL, DOCUMENT TYPES, PROGRAMMER EXPERIENCE, NUMBER OF INDEPENDENT CONSOLES, PERCENTAGE OF NEW INSTRUCTIONS	TEN MODULES ARE EMPLOYED IN THIS MODEL
MANPOWER MODEL, NORDEN	PREDICTION OF MANPOWER UTILIZATION BY PROGRAM PHASE	BOTTOM-UP BASED ON THREE VARIABLES (SEE DATA REQUIREMENTS)	PROGRAM MANPOWER REQUIREMENTS, PROGRAM LENGTH, AND PEAK MANLOADING	SINGLE EQUATION: Y = 2KATE **(-AT**2) WHERE Y = MANPOWER BY TIME PERIOD, K = TOTAL CUMULATIVE MANPOWER BY END OF PROJECT, A= SHAPE PARAMETER, AND T= ELAPSED TIME FROM START OF CYCLE
PRICE SOFTWARE COST MODEL (PRICE-S), RCA	COST ESTIMATION FOR DESIGN, IMPLEMENTATION, AND INTEGRATION TEST	TOP-DOWN BASED ON AN EXTENSIVE SET OF INPUTS; INCORPORATES SOME BOTTOM-UP FEATURES.	SEVEN CATAGORIES: PROJECT MAGNITUDE, PROGRAM APPLICATION, LEVEL OF DESIGN AND CODE, RESOURCES, PROJECT SPECIFICATIONS, RELIABILITY REQUIREMENTS, AND UTILIZATION	NEW COST ESTIMATING RELATIONSHIPS DEVELOPED FOR EACH SPECIFIC APPLICATION. PROPRIETARY MODEL.
PUTNAM MODEL	ESTIMATION OF MANYEARS, COSTS AND SCHEDULES FOR SOFTWARE PROJECTS	TOP-DOWN BASED ON FIVE VARIABLES	ELAPSED TIME, MANPOWER ON BOARD, CUMULATIVE EFFORT EXPENDED, DIFFICULTY, EXPECTED CHANGE IN DIFFICULTY	USES THE RAYLEIGH/NORDEN EQUATION TO DESCRIBE THE SOFTWARE LIFE CYCLE MANPOWER CURVE; PREDICTS CASH FLOW AND CUMULATIVE COST EITHER FOR SOFTWARE DEVELOPMENT OR THE ENTIRE LIFE CYCLE.

TABLE 1 CONTINUED

MODEL NOMENCLATURE & USER	APPLICATION	TECHNIQUES	DATA REQUIRED	COMMENTS
SOFTWARE LIFE CYCLE MODEL (SLIM)	LIFE CYCLE COST ESTIMATION AND RESOURCE CALANDERIZATION	TOP-DOWN USING DEVELOPMENT INPUTS TO COMPUTE SUPPORT COSTS	DEVELOPMENT COSTS	PARAMETRIC MODEL BASED ON PUTNAM'S APPROACH.
SOFTWARE DEVELOPMENT COST MODEL PENT CORPORATION	PREDICTS DEVELOPMENT COSTS	TOP-DOWN BASED ON 12 VARIABLES	INTERFACE COMPLEXITY, PERCENT CLERICAL INSTRUCTIONS, PERCENT INFORMATION STORAGE/RETRIEVAL, FREQUENCY OF OPERATION, PAGES OF DOCUMENTATION, BUSINESS OR NON-BUSINESS APPLICATION, NEW OR FAMILIAR COMPUTER SPECIAL DISPLAYS, RANDOM ACCESSDEVICE, PERCENT PROGRAMMERS PARTICIPATING IN DESIGN, PERSONNEL CONTINUITY, AND NUMBER OF LOCATIONS FOR PROGRAM DEVELOPMENT	COST RELATIONSHIPS ARE THE RESULT OF REGRESSION ANALYSIS ON 94 VARIABLES FOR 105 PROGRAMS
TECOLOTE MODEL, TECOLOTE REASEARCH,	DEVELOPMENT COST ESTIMATION FOR TACTICAL SOFTWARE USED IN A FIRE CONTROL SYSTEM	TOP-DOWN USING FIVE EQUATIONS	MANMONTHS, NUMBER OF INSTRUCTIONS, AIR THREATS OR SEA THREATS, COMPUTER SYSTEM SPEED, AND FAST STORAGE CAPACITY	COMPUTER SPEED AND STORAGE CAPACITY ARE THE MAJOR DRIVERS
SPREAD MODEL, WOLVERTON/TRW	ESTIMATION OF MANMONTHS AND COMPUTER USAGE ON A MONTHLY BASIS DURING DEVELOPMENT	TOP-DOWN BASED ON NUMBER OF INSTRUCTIONS AND TYPE OF SOFTWARE	COST HISTORY DATA BASE, NUMBER OF INSTRUCTIONS, DEGREE OF DIFFICULTY, AND SOFTWARE CATEGORY	SIX CATEGORIES OF SOFTWARE ARE CONSIDERED; BASIC ASSUMPTION IS THAT DEVELOPMENT COST IS PROPORTIONAL TO NUMBER OF INSTRUCTIONS; HISTORICAL DATA BASE CONSISTS OF COST PER INSTRUCTION DATA

However, more and more research is being conducted on the maintenance aspect of software cost estimation. The reason for this is becoming apparent, as it has been estimated that from forty percent to ninety-five percent of life cycle costs can be attributed to the maintenance effort [7]. The reason for this wide range of estimation seems to lie in the way various organizations view what constitutes maintenance.

The definition of software maintenance appears to vary with the organization and seems to be effected by management constraints. Software maintenance can cover the spectrum from correction of bugs caused by coding errors and design inadequacies to enhancements whose purpose is to add whole new ideas and/or design concepts not specified for inclusion in the original system. The lack of a standard definition for maintenance is a major contributor to the paucity of data collection in this area. In many organizations, especially military, as top level management personnel rotate through specific positions, different definitions of what constitutes software maintenance also rotate through those positions and the organizational levels they control. As a direct result, data collection requirements change to complement the definition of maintenance and, as a consequence, no consistent track of a project's manpower usage history can be recreated. Of greater significance is the lack of a standard maintenance policy within the organization to include a maintenance strategy which will add to the degree of software maintainability, if not assure it.

In view of the large costs associated with software maintenance, GAO conducted a study which reviewed fifteen Federal computer installations in detail. Their findings pointed to two major contributors to the problem; the fact that, in the majority of agencies, maintenance is not managed as a separate, identifiable function, and there is an absence of a uniform definition of maintenance [8]. GAO's recommendations included development of a standard definition of maintenance by the National Bureau of Standards and delineation of maintenance as a discrete function by agency heads. In the interim, GAO developed a checklist of items, the consideration of which could reduce maintenance costs. In the checklist is a set of categories for recording maintenance costs. These six categories appear to reflect GAO's definition of maintenance and as such, are listed below:

1. Modify or enhance software to make it do things for the end user that that were not requested in the original system design.
2. Modify or enhance software to make it do things for the end users that were called for in the original design but which were not present in the first production version of the software.
3. Remove defects in which the software does something other than what the user wanted ("does the wrong things").
4. Remove defects in which the software is programmed incorrectly ("does the desired calculation, but gives an incorrect answer").
5. Optimize the software to reduce the machine costs of running it, leaving the user results unchanged.
6. Make miscellaneous modifications, such as those needed to interface with new releases of operating systems.[9]

This "definition" appears to have general applicability over the broad spectrum of activities which can be and have been grouped under the category of software maintenance. However, number one may cause problems in the context of maintenance cost estimation techniques based on the Rayleigh curve. Since enhancements necessarily require some design/development effort by their very nature (they give the product capabilities not called for in the original design), the manning level in such effort would exhibit a rise and then a fall in magnitude in the Rayleigh fashion, thus creating a series of small Rayleigh curves within the maintenance phase. As long as this behavior did not vary greatly from the normal maintenance effort for that project, it would not have much effect on the project. However, if the front end of the curve rose beyond some predefined maintenance support boundary, then it would indicate the presence of a full scale development project instead of a pure maintenance effort, and it should signal the completion of the old project and the start of a new one. Therefore, because of the nature of the software life cycle, even a standard definition of maintennace has grey areas and management judgement must be used in its application.

The GAO definition does, as stated earlier, provide a good, general definition of software maintenance and, as such, for the purposes of this thesis, software maintenance encompasses all of its categories.

B. PROBLEM DEFINITION

James P. Green and Brenda F. Selby, formerly of the Naval Postgraduate School, having reviewed Putnam's Software Cost Estimating Model, the Army Macro-estimating Model, the Lehman-Belady Model, and the Parr Model, have proposed a dual theory for maintenance requirements estimation. They proposed that, if one considered maintenance to include all effort applied to a software project from the time that the product was released to the user, that the peak maintenance manloading required could be calculated by computing the inflection point on a Rayleigh curve for the total software life cycle effort. They further predicted that one could predict the minimum maintenance manloading requirments by computing the inflection point on the Rayleigh curve representing the maintenance life cycle.

The proposed Green/Selby Model, upon cursory examination, appears to have tremendous potential as a tool for the manager of software projects. However, Green and Selby were not able to obtain sufficient data to thoroughly validate the applicability of the model to real world situations. Therefore, much further work is needed in this area.

C. RESEARCH OBJECTIVES

The objectives of the research are twofold: to evaluate the Green/Selby model for prediction of maintenance costs via projection of maintenance manloading, both for maintenance team development and for outyear support resource estimation, and to provide an analysis of applications of the model in areas other than project management and control. The Green/Selby model addresses two areas, a maintenance planning concept which is concerned with the overall maintenance strategy as applied to a particular software project and a maintenance control concept which is concerned with manloading requirements estimation. Only the latter will be dealt with in this research.

The evaluation of the model will be accomplished in the pursuit of three subobjectives. The first is to provide an analysis of software maintenance costing problems and a synopsis from the literature of other existing models and techniques, some of which were used in the initial Green/Selby model development, and some of which the authors feel are of equal importance and which may contribute to further development or application of the Green/Selby model.

The second subobjective is to validate the development of the Green/Selby model through analysis of the mathematical relationships and through recreation of the empirical development. The third subobjective is to validate the model with actual data from as many different sized software projects as possible to ascertain the degree to which the model is applicable to real world software costing problems.

Based on the results of the data analysis, projections will be made as to possible applications of the model in areas other than cost estimation, if such applications appear to exist.

D. ASSUMPTIONS/LIMITATIONS

Three major assumptions were made at the onset of the research effort for this thesis. Other assumptions were necessary at specific junctures of the research but they do not apply in every case, so they are discussed where they are applicable. The major assumptions are as follows:

1. It was assumed, based on limited prior study in the subject area, that the software project life cycle and all of its phases followed the general pattern of the Rayleigh curve.
2. It was assumed that the Green/Selby Model was valid in its development though not thoroughly tested in its application.
3. It was assumed that there is little difference in how project size affects the manning behavior of a project during the individual phase cycles and during the total project life cycle.

Three major constraints were found to limit the research effort. They are as follows:

1. There was found to be a serious lack of readily available data which applied to the maintenance phase.

2. There appears to have been little major research done in the area of software maintenance manloading/cost estimation.
3. Because of the nature of the subject area and the variance of maintenance data collection across organizations, the research completed and data collected to date appears to have involved what are recently being categorized as inefficient and maintenance-intensive design techniques. Therefore, the applicability of early works and present research using old data may become suspect, if not invalid, by the use of such techniques as modularization, information hiding modules, and the use of other, recently developed, software tools. Hence, the new methods may alter the old relationships entirely.

E. RESEARCH METHODOLOGY

The research methodology implemented by the authors of this thesis was fivefold, to include literature search, data search/collection, research design, model validation, and data analysis/evaluation.

A literature search was conducted both by manual and automated means. A manual search produced most of the references, used by Green and Selby, which were used to provide the researchers with a solid background in the area of study and to recreate, as closely as possible, the knowledge base from which the Green/Selby model was developed. Two automated searches were conducted, one through the Defense Logistics Information Studies Exchange(DLSIE) and one via the computerized library search network. Both searches produced numerous writings of interest from the private and military sectors.

The search for data highlighted the largest single stumbling block to research in the area of software maintenance, that of a lack of adequate data collection by maintaining activities. Actual manloading records have usually been kept during the development phases of numerous software projects; however, maintenance data appears to have been recorded only recently, and then only sporadically at best. The search for data was conducted successfully via telephone conversations with the following persons/organizations;

Goddard Space Flight Center, Greenbelt, Md.; and

Dr. Willa Kay Weiner-Ehrlich, consultant, Bankers Trust Co., NY, NY.

The following organizations were contacted in the course of the search with no significant results:

Data And Analysis Center for Software, Griffis AFB, NY;

United States Army Computer Systems Command, Ft. Belvoir, Va.;

Aeronautical Systems Division, Wright Patterson AFB, Dayton, Ohio; and

Data Systems Design Center, Gunter AFSTA, Montgomery, Ala.

Valuable support and/or referral information were received from the following persons:

Dr. Robert Grafton, Office of Naval Research, Washington, D.C.;

Dr. Victor Bascili, University of Maryland, College Park, Md.;

Mr. David Weiss, Naval Research Laboratory, Washington, D.C.;

Ms. Cheryl Maloney and Mr. Robert Jones, United States Army Computer Systems Command, Ft. Belvoir, Va.; and

Mr. Lawrence Putnam, Quantitative Software Management, Inc., McLean, Va.

The NASA SEL data base, which contains data on about forty software projects, was received from the Data and Analysis Center for Software, but it was discovered that maintenance data is just now being collected, and no significant aggregate will be available for approximately two years.

A report, produced for the Air Force by General Research Corporation of Santa Barbara, Ca., indicated that the Planning and Resource Management Information System (PARMIS) at the Air Force Data Systems Design Center (AFDSDC), Gunter AFSTA, Montgomery, Ala., held a large, relatively untapped, data base of manpower usage (projected and actual) from about 2000 projects. However, the data search revealed that PARMIS was replaced by a new Personnel Cost/ Accounting System in 1977/1978 and it appears that the former data base was deleted due to format incompatibilities with the new system.

As such, it is apparent that little maintenance data is available or, if in existence, it is very difficult to locate.

Once a knowledge base was developed and data collected, the research process was begun. That process is listed in general:

A. Develop mathematical relationships in terms of equations;

B. Validate Green/Selby model development;

C. Analyze empirical project data in terms of Green/Selby model; and

D. Interpret data analysis.

In order to attempt to validate the Green/Selby model, the model development was recreated as closely as possible using the same or similar data.

Data analysis was conducted by using various non-linear curve fitting techniques to fit actual life cycle man-loading values to the Rayleigh model. Then, Green/Selby model relationships were calculated and plotted against maintenance phase values. The above techniques allowed evaluation of applicability of the Green/Selby model with actual project data.

F. OVERVIEW OF THE THESIS

In this introductory chapter, the term software 'maintenance' was defined and its importance in the context of the data systems organization was discussed. The problem to be considered in this thesis has been presented and the objectives of the research effort intended to resolve the problem have been delineated. Assumptions made at the onset of the research effort and major limitations encountered during the course of the research were discussed. Finally, the research methodology was outlined. Chapter II looks at various models and cost estimating techniques which were used as a basis for the development of the Green/Selby model. It also includes a synopsis of other models which the researchers feel are of importance to the particular area of study. Chapter III presents an in-depth analysis of the Green/Selby model, and its proposed applications. Chapter IV provides a mathematical and empirical validation of the model, using similar data to that used by Green and Selby originally. Chapter V discusses the data analysis, and thus, the empirical model validation evaluation. Finally, Chapter VI summarizes the thesis and presents conclusions and recommendations.

II. SOFTWARE MAINTENANCE COST ESTIMATION MODELS

A. CURRENT TECHNIQUES USED AS A BASIS FOR THE GREEN/SELBY MODEL

1. Putnam's Software Cost Estimating Model

Putnam developed his method for software cost estimation by studing various systems designed by the United States Army Computer Systems Command (USACSC) and comparing them to the Rayleigh life cycle profile developed by Peter V. Norden in the 1960's. This life cycle profile, depicted in Figure (2.1), linked the individual cycles of each of the life cycle phases and added them together producing the profile for the entire project. Putnam's empirical studies showed that, for the system studied, the software life cycle

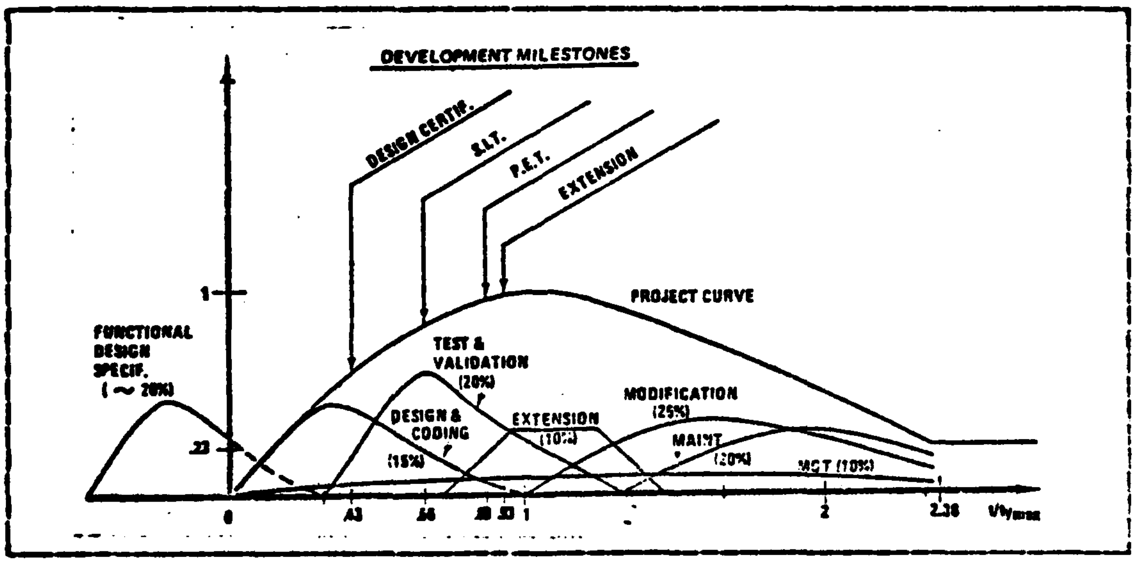


Figure 2.1 Rayleigh Project Life Cycle Profile

exhibits a rise in manpower up to a peak and then a trailing off portion corresponding very well with Norden's Rayleigh curve.

Putnam attempts to answer the questions "How do I know how long a software project will take, and how much will it cost"? [10] In order to do this, Putnam analyzes the following areas:

- •Optimum Man-loading over life cycle
- •Total Manpower over life cycle
- •Cost per year
- •Life Cycle cost in
    - •Current $
    - •Inflated $
    - •Discounted $ (for E. A.)
- •minimum $ benefits to break even over economic life
- •Risk profiles for:
    - •Manpower
    - •Costs
    - •Project completion [11]

The Rayleigh model for cumulative manpower utilization, used by Putnam, is given by the formula

$$Y = K(1-e)^{-at^2} , \qquad (2.1)$$

where

Y = cumulative manpower used,

K = the total number of man-years of life cycle effort,

a = the curve shape parameter, and

t = the elapsed time in years.

However, the most popular form of the curve is the derivative form for current manpower utilization expressed by

$$Y' = 2Kate^{-at^2} . \qquad (2.2)$$

Empirically derived:

$$a = 1/2t_d^2 , \qquad (2.3)$$

where
t_d = the time to reach peak effort.

In terms of software projects, t_d has been empirically shown to correspond very closely to the design time (or the time to reach initial operational capability) of a large software project [12].

With t_d representing the development time for the system, equation (2.3) can be substituted into the Rayleigh equation, and the shape of the curve, together with the accompanying equation, allow us to project what the manpower requirements and cash flow for system development will be at any given time. (Cash flow is calculated by multiplying manpower projections by the current personnel salaries.) The equation representing this curve is[13]

$$Y' = K/t_d^2 te.^{-(t^2/2t_d^2)} \qquad (2.4)$$

Putnam found that there was a fundamental relationship in software development between the number of source statements in the system and the effort, development time, and the state of technology being applied to the project. The equation that describes this relationship is :

$$Ss = Ck\, K^{1/3} t_d^{4/3} , \qquad (2.5)$$

where

Ss = the number of end product source lines of code delivered,

K = the life cycle effort in man-years,

t_d = development time, and

Ck = a state of the technology constant.

At least three different estimates of program size should be made before development of the system begins. They should be made once during the system definition phase and at least twice during the functional design and specification phase. This will insure a very realistic estimate of the size of the system. Admittedly, estimation of Ss and Ck are extremely difficult; however, if similar projects have been done in the past their values should remain fairly constant.[14]

Putnam's model seems to work extremely well with large scale software projects but it does not seem to fit well for projects under 10,000 lines of source code [15]. The largest problem with the use of Putnam's model is the reliance on past experience and historical data banks, if in fact they exist, to estimate the size and complexity of the current project. It also pays little attention to operation and maintenance costs after development is complete or non-manpower related items such as computer time and travel allowances which may influence total life cycle costs to a great extent.

2. Parr's Software Cost Estimating Model

The Parr model was developed by F. N. Parr after he had studied the work done by Norden and Putnam on the Rayleigh curve. Parr was concerned that the Rayleigh curve failed to answer questions about the learning curves usually associated with the start of new projects. He also felt that it made the assumption that the skill available for a project depends on resources which have been applied to it. This, he states, confuses the intrinsic constraints of the linear learning curve with the rate at which software can be written, based on management's economically governed choices in response to these constraints. Parr further states that:

The process generally used to develop new software can be thought of as the successive solution of a large number of small problems. The solution of each of these individual problems is a decision which defines some feature of the final program. A development project corresponds to starting out with some fixed bounded set of problems to be solved and ending with enough decisions having been made for a working product to be available.[16]

Parr utilized a binary tree concept to statistically determine the number of possible problems and decided that the proportion of the problems solved at time t, denoted as W(t), was given by the formula

$$W(t) = 1/(1 + A e^{-at}), \qquad (2.6)$$

where

A = a constant, and

a = shape parameter.

By solving this equation, he could determine the expected change in the size of the visible unsolved node set as a linear function of the work completed. The importance of this was that he determined that the rate at which work could be usefully input to the development process was proportional to the size of the set of visible unsolved problems, V(t). He further determined that when the optimal input effort is applied, steps in the development would be achieved at a rate proportional to V(t). Thus the work-rate could be determined by solving for V(t) which he developed into the equation :

$$V(t) = (1/4) \operatorname{sech}^2 ((at + c3)/2), \qquad (2.7)$$

where

c3 = an integration constant.

Figure (2.2) shows the resulting curve overlayed on a corresponding Rayleigh curve.

It can be seen that the back portion of the sech-squared function correlates very highly with the Rayleigh

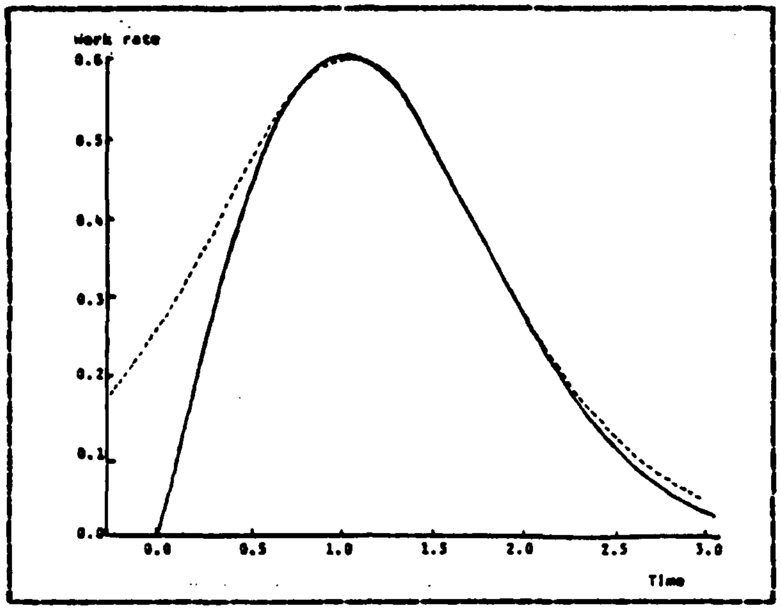


Figure 2.2 Comparison of Sech² and Rayleigh Curves

curve. However, the front portion does not show a well-defined starting point, as is the case with the Rayleigh curve. Parr feels that the front portion of the curve represents that portion of the work done before the official starting date for a project. He feels that this is more realistic than the Rayleigh curve.

Parr went on to explore the complexity factors introduced by the increased usage of structured programming and developed the formula:

$$V(t) = [aAe^{-2at} / (1 + Ae^{-2at})^{3/2}]/a. \qquad (2.8)$$

The resulting curve has its peak shifted slightly to the right of the sech-squared function; which predicts that peak work rate will occur after half the project has been done. This he asserts is in keeping with the theory that design may be slower, but there will be a compensating reduction in testing and maintenance effort.

3. Army Macro-estimating Model

Having already developed a number of software systems, the Army decided that it needed a method which would be simple, effective, and reasonably accurate for determining and controlling manpower and dollar resources for any point in the software life cycle.

After reviewing the data on its existing systems, the Army chose the mathematical relationship developed by Norden where:

$$Y' = 2Kate^{-at^2} \qquad (2.9)$$

This equation was the same one used by Putnam, and it was used by the Army to derive the various milestones to be used by system managers. By comparing the actual resources used when these milestones were reached, the action officer could take corrective action if, statistically, those resources used were outside the control limits.

These milestones were developed based on step-by-step procedures given in the following cases:

<u>Case I</u>: <u>System already under development</u> (<u>resources budgeted</u>).

Using budget data, the maximum level of manpower (Y'_{max}) and the number of years to reach maximum effort ($t_{Y'max}$) is determined. Rather than compute the values for outyear manpower loading, Table II is used to compute the values of Y' for the appropriate $t_{Y'max}$. By multiplying any entry opposite its time period by K, the appropriate number of manyears are obtained. The units of K and t will determine the dimensions.

<u>Case II</u>: <u>New system</u> (<u>no resource data</u>).

Total man-years of effort and peak time for manpower loading is derived using Bayes' theorem. Based on empirical

TABLE II

Ordinates for Manpower Functions

t	$t_{Y'max}$	1	2	3	4	5	6	7
0	a	.50	.1250	.0556	.0310	.0200	.0139	.0120
1		.60653	.22062	.10510	.06057	.03920	.02739	.02020
2		.27067	.30326	.17794	.11031	.07384	.05255	.03918
3		.03332	.24349	.20217	.14153	.10023	.07354	.05585
4		.00134	.13533	.18271	.15163	.11618	.08897	.06933
5		.00001	.05492	.13852	.14307	.12130	.09814	.07906
6			.01666	.09022	.12174	.11682	.10108	.08480
7			.00382	.05112	.09461	.10508	.09845	.08664
8			.00067	.02539	.06766	.08897	.09135	.08497
9			.00009	.01110	.04475	.07124	.08116	.08036
10			.00000	.00429	.02746	.05413	.06926	.07356
11			.00000	.00147	.01567	.03912	.05691	.06530
12				.00044	.00833	.02694	.04511	.05634
13				.00012	.00413	.01770	.03453	.04729
14				.00002	.00191	.01111	.02556	.03866
15				.00000	.00082	.00666	.08130	.03081
16				.00000	.00033	.00382	.01269	.02395
17					.00012	.00210	.00853	.01817
18					.00004	.00110	.00555	.01346
19					.00001	.00055	.00350	.00974
20					.00000	.00026	.00214	.00689

data from internal systems, a probability versus K density function was derived without regard to type of system. Further analysis determined frequency of system type and probability of occurence of each type. Using estimates based on past USACSC experiences (the average K value for all systems under development and average K for the functional type of system), initial estimates for a new development are calculated from regression graphs. Then, applying Bayes' theorem to average these individual estimates in the weighted probability sense yields a better estimate of K with a smaller standard deviation (i.e. better confidence in the estimate). To improve estimates and reduce uncertainty, Bayes' theorem is successively applied.[17]

4. The Lehman-Belady Model

L. A. Belady and M. M. Lehman developed their model by studing the management and evolution of the OS/360 operating system. They felt that this system gave them a good view of the processes and managerial thinking that goes into the development and programming of medium to large-sized projects. The decision to use this system was reached after they had surveyed a number of versions and releases of OS/360 before their study began. The data for each release included measures of the size of the system, the number of modules added, or changed, the release date, information on manpower used, machine time used and costs involved in each release. In general, there were large, apparently stochastic, variations in the individual data items from release to release.

The data exhibited a general upward trend in the size, complexity, and cost of the system and the maintenance process. This was indicated by comparing the components, statements, instructions, and modules handled over the system life cycle. The various parameters were averaged to expose trends. When averaged, previously erratic data appeared to become strikingly smooth, displaying nonlinear - possibly exponential - growth and complexity.

As a result of their research, they postulated three laws of Program Evolution Dynamics.

I. Law of continuing change. A system that is used undergoes continuing change until it is judged more cost effective to freeze and recreate it.

Software does not face the physical decay problems that hardware faces. But the power and logical flexibility of computing systems, the extending technology of computer applications, the ever-evolving hardware, and the pressures for the exploitation of new business opportunities all make demands. Manufacturers, therefore, encourage the continuous adaptation of programs to keep in step with increasing skill, insight, ambition, and opportunity. In addition to such external pressures for change, there is the constant need to repair system faults, whether they are errors that stem from faulty implementation or defects that relate to weaknesses in design or behavior. Thus, a programming system undergoes continuous maintenance and development, driven by mutually stimulating changes in system capability and environmental usage. In fact, the evolution pattern of a large program is similar to that of any other complex system in that it stems from the closed-loop cyclic adaptation of environment to system changes and vice versa.

As a system is changed, its structure inevitably degenerates. The resulting system complexity and reduction of manageability are expressed by the Second Law of Program Evolution Dynamics.

II. Law of increasing entropy. The entropy of a system (its unstructuredness) increases with time, unless specific work is executed to maintain or reduce it.

This law too expresses vast experience, in part by data...This, in turn, leads to the formulation of the Third Law of Program Evolution Dynamics.

III. Law of statistically smooth growth. Growth trend measures of global system attributes may appear to be stochastic locally in time and space, but, statistically, they are cyclically self-regulating, with well-defined long-range trends.

The system and the metasystem -the project organization that is developing it- constitute an organism that is constrained by conservation laws. These laws may be locally violated, but they direct, constrain, control, and thereby regulate and smooth, the long-term growth and development patterns and rates. Observation, measurement, and interpretation of the latter can thus be used to plan, control, and forecast better the product of an existing process and to improve the process so as to obtain desired or desirable characteristics.[18]

Having postulated these three laws, they commenced the process of defining a complexity factor C(R) for the various program releases, each of which were assigned Release Sequence Numbers (RSN's). From the available data they proposed the formula:

$$C_R = MH_R / M_R, \qquad (2.10)$$

where

$M(R)$ measures the size of the the system in modules and

$M(HR)$ records the number of system modules that have received attention.

Utilizing this complexity factor, they stated that the design - programming - distribution usage system has a feedback driven and controlled transfer function and an input-output relationship. This feedback results, sometimes, from constant pressure to supplement system capability and power. This constant pressure normally results in work pressures building up as growth rate increases. Accordingly, the growth rate increases the size and complexity of the system and reduces the quality of design, coding, and testing. This is accompanied by lagging documentation, and other factors, which emerge to counter the increasing growth rate.

Eventually, the above relationship resulted in the need for a system consolidation in which correction, restructuring, and rewriting were done with few, if any, functional enhancements. The consolidation often results in the shrinking of a system during such a release, rather than the growing normally experienced with each new release. This, they observed, occurred with every twenty to twenty-one releases of the system. They further observed that successful releases appeared to have an upper bound of about 400 modules.

Since the majority of managers base their decisions on available budgets, Lehman and Belady proposed that the total expenditure for all activities involved with the project be equal to the budget, and hence, the formula for the budget (B) is given by:

$$B = P + A + C \qquad (2.11)$$

where

P is units of fault extraction activity termed progressive,

A is the amount of resources associated with documentation, administration, communication, and learning activity termed antiregressive.

C is the increasing work demanded to cope with the neglect of A, and is given by the formula

$$C = \int_0^t (1-m)\, kP\,dt, \text{ and} \tag{2.12}$$

where

m and k are defined below.

The formula for antiregressive activities is:

$$A = mkP \tag{2.13}$$

where

m is the management factor, which is the fraction of progress, kP, that is actually dedicated by management to A activity, and

k represents the inherent A activity required for each unit of P activity so that complexity does not grow and is given by the formula

$$k = A / P. \tag{2.14}$$

Management is assumed to have full control of the allocation of its resources and the division of effort between P- and A-type activities. Management cannot, however, directly control the growth in complexity that accumulates, except by utter concentration on complexity control through restructuring. This is an activity that is strictly antiregressive and, as such, is psychologically difficult to inspire, since it yields no direct, short-term, benefits. [19]

An interpretation of their model suggests that more rapid work leads to greater pressures on the team, and hence more errors. This, in turn, requires greater repair activity. However, the data indicates that this problem is mainly incurred in the same release rather than discovered and undertaken thereafter. Futhermore, since it appears to lead to an increase in the fraction of the system handled, it suggests that the maintenance teams tend to remove the symptoms of a fault rather than to locate and repair its cause. This problem is reduced through proper communication, documentation, and learning by the programming team.[20]

B. OTHER MODELS OF INTEREST

1. Jensen Model

Randall W. Jensen [21] stated that, because traditional intuitive estimation methods consistently produce optimistic results which contribute to the too familiar cost overrun and schedule slippage, customers for software products are becoming less willing to tolerate the losses associated with inaccurate estimates. He, therefore, derived his model based primarily on the work done by Norden, Putnam, and Doty Associates.

In conjunction with the familiar Rayleigh equation

$$Y' = 2Kate^{-at^2} \qquad (2.15)$$

Jensen's model consists of a series of equations for system productivity, initial project staffing rate, system complexity, system size, development effort, and risk analysis.

He defines the productivity relationship by the equation:

$$\overline{PR} = C_n (K/t_d^2)^{-B}, \qquad (2.16)$$

where

$\overline{PR}$ = average project productivity (source lines per year),

K = Total life cycle cost in manyears,

t_d = development time in years and is defined as the peak time for the Rayleigh curve,

C_n = a proportionality constant, and

B = slope of productivity relationship.

While this equation is not actually related to the system difficulty, it is related to the rate at which staff is applied to the task. Intuitively, productivity is an inverse function of the number of people directly involved with a development task due to the associated losses caused by the number of communication paths in the organization. This phenomenom can be accounted for by utilizing the relationship

$$M = K/t_d^2, \tag{2.17}$$

which is the formula for the initial project staffing rate, M, and is extremely important in determining the optimum project staffing rate.

Most, if not all, of the projects studied by Jensen, appeared to demonstrate a consistent pattern which could be used to classify each project into distinct categories. These categories were dependent on the interface complexity, logical complexity, and the percentage of new development in the system, all of which seemed to be defined by the ratio

$$K/t_d^3. \tag{2.18}$$

The expression K/t_d^3, in a practical sense, represents a natural equilibrium between the lifecycle cost and development time for a specific class of software projects. As a result, similar projects tended to maintain this equilibrium so that as the system size increased, the development schedule increased correspondingly. This equilibrium also maintained the staffing rate,

$$K/t_d^2, \tag{2.19}$$

within bounds that could be effectively accommodated by the project. Thus, he used this equilibrium expression to define system complexity (D) as

$$D = K/t_d^3. \tag{2.20}$$

The value of D can be thought of as a limiting parameter in determining the minimum development time that an organization can achieve for a given software project. Table III shows the values of D determined by Jensen from Putnam's analysis of USACSC data.

The next equation, developed by Jensen, was referred to as the software equation, relating the size of the system to the technology being applied by the developer in the implementation of the system. In deriving this equation, Jensen utilized an extension of the productivity relationship proposed by W. F. Sampson of General Electric Company.

Sampson [22], after reviewing data supplied by Putnam from 19 USACSC projects, determined that only a subset of these projects represented a consistent development environment and were sufficiently documented to be of value in establishing the model parameters. Evaluation of this refined set of data obtained a B value of -0.50 for the basic relationship between productivity and project stress instead of the -0.667 obtained when all the data was used.

With Sampson's work in mind, Jensen derived the software equation to establish the rate of source code development, dSs/dt. In his development, he assumed that the portion of the project effort devoted to code production, P1(t), was characterized by a Rayleigh curve, which was complete at td.

TABLE III

Project Complexity Values

Value	Characteristics
8	Applies to new systems with significant interface and interaction requirements within a larger system structure. Operating system and real time processing developments with large percentages of logical code are typical of this class of systems.
15	Applies to new standalone systems developed on firm operating systems. The interface problem with the underlying operating system or other parts of the system is minimal. New applications software is typical of this class of systems.
27	Applies to complete rebuilds of existing standalone systems where major portions of existing logic can be used.
55	Applies to composite systems where existing systems are combined or integrated with little or no modification of existing software.

Then if

$$t_d/t_d1 = 6, \tag{2.21}$$

where

t_d1 = the time of peak manloading on the Rayleigh curve, coincidental to development time, and

$$\int_0^{t_d} P(t)\,dt = \int_0^{t_d} (K/t_d^2)\,t\,e^{-(3t^2/t_d^2)}\,dt = 0.95K/6, \tag{2.22}$$

then the burdening rate for this project is

$$\frac{\int_0^{t_d} P(t)\,dt}{\int_0^{t_d} P1(t)\,dt} = \frac{0.3934K}{0.95K/6} = 2.49, \tag{2.23}$$

where

P(t) = staffing level. The rate of source code development, dSs/dt, is assumed to be proportional to the rate of code production, P1(t) so that

$$Ss = 2.49\ \overline{PR}\ P1(t),$$

and

$$Ss = 2.49\ \overline{PR}\ K/t_d^2 \int_0^\infty t e^{-(3t^2/t_d^2)} dt \qquad (2.24)$$

$$= 2.49\overline{PR}\ K/6.$$

Substituting the empirically derived value of $\overline{PR} = C_1 M^{-0.5}$ gives:

$$Ss = (2.49C_1/6)K^{.5} t_d,$$

or

$$Ss = C_t \sqrt{K}\ t_d, \qquad (2.25)$$

which is the software equation where

C_t = a developer technology constant.

This technology constant, Ct, is a factor, or constant of proportionality, that allows the user to relate the system size, Ss, the life cycle effort, K, and the development time, t_d, for any specified project. The constant accounts for all variations in the life cycle effort for projects which have similar size and schedule properties. The constant is then a measure of the developer's production technology, or ability to implement the project. This includes such factors as the availability of computing resources, organizational strategies, development tools and methodologies, familiarity with the target computer, etc.

The technology constant considers two aspects of production, the environmental aspect and the technical aspect. The environmental aspect includes those factors dealing with the basic computing environment. The environmental factors determine a technology constant which normally ranges between 2000 and 5000, with higher values characteristic of higher productivity environments; ie., from primitive tools to dedicated advanced tools and resources. The technical aspects of the technology constant are accounted for through the use of adjustment factors applied to the basic technology constant by use of the formula

$$C_t = C_{tb} / \sqrt{\sum_{i=1}^{14} f_i} = C_{tb} / f_t, \qquad (2.26)$$

where

C_{tb} = basic technology constant,

f_i = ith adjustment factor, and

f_t = total adjustment factor.

The adjustment factors include those effects which are beyond the basic development environment and are project specific. The factors, which are shown in Table IV, are examples of those found in a command and control system environment.

Feeling that his model could be understood better as a linear programming problem presented in a graphical format, Jensen defined the additional formulas which he could use for this forum. The first formula was for the development effort (E) which he derived as:

$$E = \int_0^{t_d} P(t)\,dt = 0.4K. \qquad (2.27)$$

TABLE IV

Technology Constant Adjustment Factors

Factor		Value	
Number	Description	Yes	No
1	Special display requirements	1.11	1.00
2	Detail operational requirements	1.00	1.54
3	Changes to operational requirements	1.05	1.00
4	Real time operation	1.33	1.00
5	CPU memory constraint	1.25	1.00
6	CPU time constraint	1.51	1.00
7	First software developed on CPU	1.92	1.00
8	Concurrent ADP hardware development	1.67	1.00
9	Developer using computer at another facility	1.43	1.00
10	Development at operational site	1.39	1.00
11	Development computer different than target computer	2.22	1.00
12	Development at multiple sites	1.25	1.00
13	First use of language	1.80	1.00
14	MIL-STD documentation	1.40	1.00

The next was a relationship (R) determined by the system size and the developer's approach to the project and was given by:

$$R = Ss/C_t = \sqrt{K}t_d \ . \qquad (2.28)$$

Then, utilizing the formulas for M and D, equations (2.17) and (2.20), where M represents a fixed staffing rate or management stress curve, and D represents projects of fixed complexity, he could plot all these equations on a solution surface for various size projects as shown in Figure (2.3).

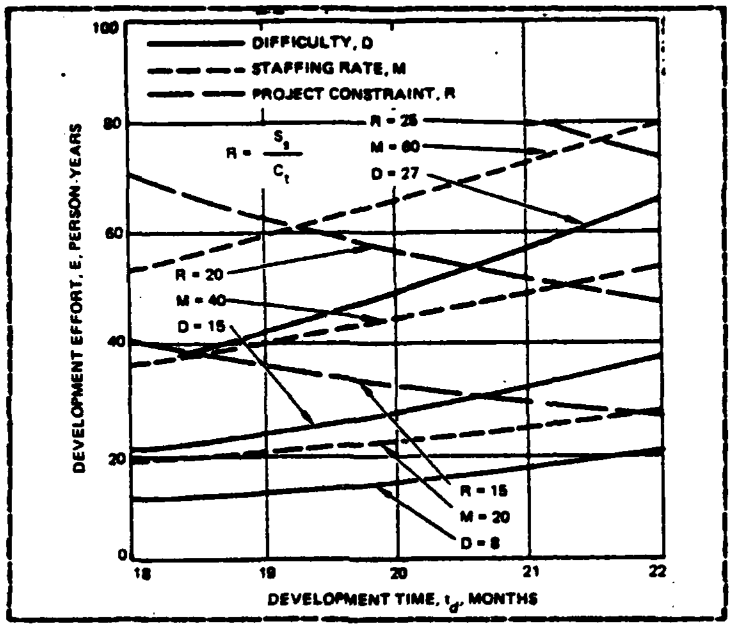


Figure 2.3 Macro-Estimating Model Solution Surface (Graphical Representation)

For example, assume a project is defined with a value, D = 15. for implementation considerations, a project can be treated as being more complex than it actually is, but it can never be treated as being simpler; thus, the region defined by $0 < D \leq 15$ must be considered as a feasible solution space. The same type of reasoning can be used with the parameters Ss and Ct in the R curve, so that if the ratio R = Ss/Ct =30, is specified, the feasible region, $30 \leq R < \infty$, is defined. Similarly a value of M = 20 defines a region $0 < M \leq 20$, since less staff can always be used than is available, never more. The intersection of the R and D curve or R and M curve determines the minimum development time, depending on the resulting feasible region mapped by three curves. Secondary constraints of development time, td, and development effort can also be used to define the feasible solution region.[23]

With respect to either effort or time, the optimum solution will be located at one of the vertices defined by the constraint lines. The possibility exists that, once all the constraints, D, R, M, E, and t_d, are plotted on the solution surface as shown in Figure (2.4), some of the constraints will be eliminated from futher analysis by the manner in which other constraints intersect to form the bounded region. If the constraints bound a null region, either the cost or schedule is too optimistic and cost or schedule overruns in software development are likely to occur. However, by utilizing the values for K and t_d

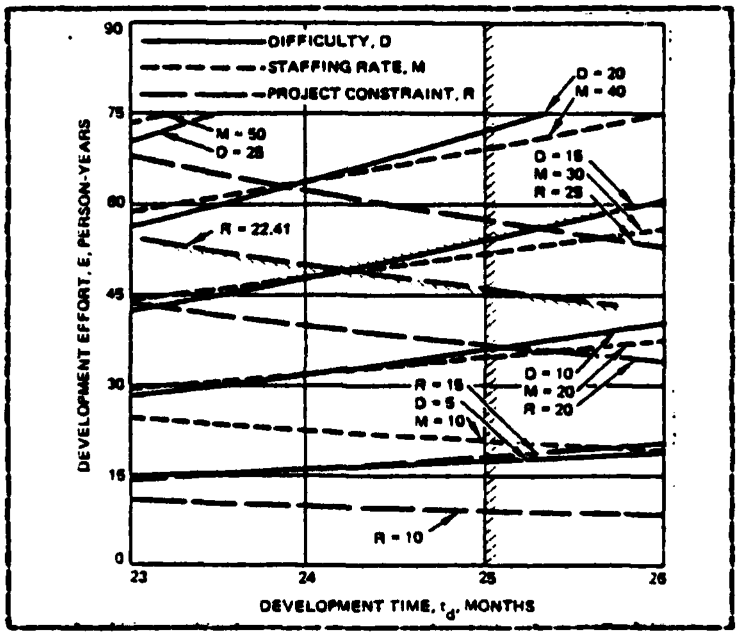


Figure 2.4 Feasible Solution Region

obtained from the graph and substituting into the Rayleigh equation, the optimum staffing profile (Y') can be obtained.

Recognizing that the calculations made by the model assume that the input parameters are exactly known, and that there is a degree of uncertainty associated with each of the input parameters, Jensen postulated, for risk analysis, that the deviation from the mean can be calculated using the relationship

$$\sigma f = [(\partial f/\partial Ss)^2 \sigma_s^2 + (\partial f/\partial C_t)^2 \sigma_c^2 + (\partial f/\partial D)^2 \sigma_D^2]^{0.5},$$

where

$$f = t_d = \sqrt[5]{(Ss/C_t^2)\ 1/D},$$

or

$$f = K = [(Ss/C_t)^3 D]^{0.4}. \qquad (2.29)$$

Similar expressions for f could be found by using M, instead of D, as the bounds for the feasible region. In cases where both M and D interact, the expression for f should be considered invalid and no alternative solution was provided.[24]

As an example of this risk analysis technique he provided the example where Ss = 55,642; D = 15; s = 2,058; σ_D = 1; and t = 0.482. The results were then plotted as shown in Figure (2.5). The results show that the probability of meeting the required schedule is 94 percent.[25]

2. Other Models

A description of some additional models which were not used in this thesis but the reader might find informative are provided in Appendix A and Appendix B, as described by R. Thibodeau and R. W. Wolverton, respectively [26,27].

C. CHAPTER TWO SUMMARY

The thesis of the models used in this chapter and in others that were found in the literature, was to try and give management a tool with which they could predict the cost of software, the time for producing this software, or

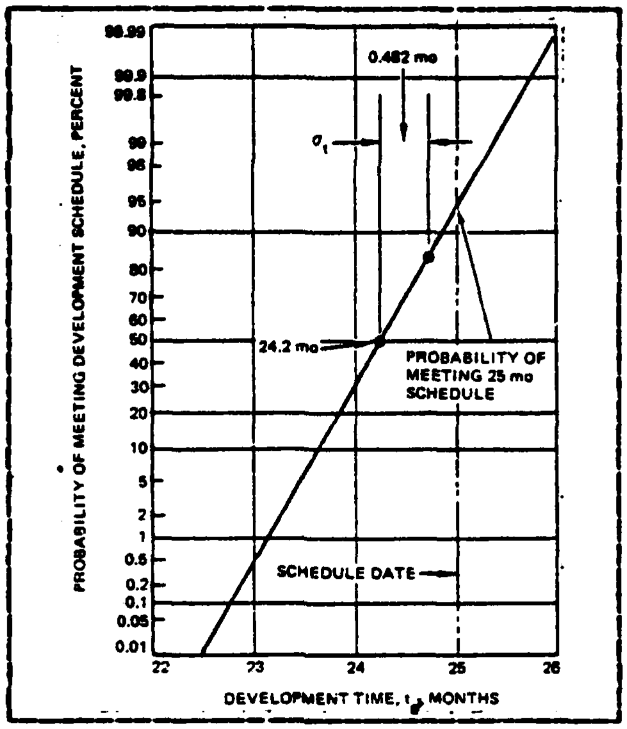


Figure 2.5 **Risk Analysis of Schedule Using Graphical Technique**

both. Most, if not all of the models require the use of historical data and/or management's previous experience as a portion of the predictive process.

It was Putnam's view that software production followed a Rayleigh curve. This curve, he asserted could be calculated utilizing historical data to determine the technology constant (Ck), and the estimate of source lines of code for this type of project (Ss), plus the budgeting information for the total number of man-years for the systems life cycle.

The Army Macro model utilized Putnam's technique, but, at various time increments, would compare actual results with those predicted and, if the actual resources expended were statistically outside some preset control limits, corrective action would be taken.

Parr felt that Putnam's model did not take into account the effort that was completed prior to the actual starting date. He, therefore, proposed a model which would take this work into account in the early part of the project. It also correlated well with the work done by Norden and Putnam with the Rayleigh curve, both at the peak level and in the later stages.

Lehman and Belady found in their study of the evolution of the OS/360 operating system programming effort that, as the size and complexity of each release which contained functional enhancements increased, so did the number of errors and, thus, the amount of maintenance effort also increased. Therefore, they postulated that for any system there is a time when it is better to restructure and consolidate than to continue with additional enhancements.

Jensen felt that Putnam's model required some expansion and refinement. This he attempted to accomplish through the use of linear programming and graphical representation of his results.

III. MAINTENANCE COST ESTIMATING VIA THE GREEN/SELBY MODEL

The Green/Selby model includes two techniques: the first characterized by a macro approach and the second by a micro approach. The results of the application of both techniques to project planning parameters are compared and then weighed against managerial and organizational constraints to analyze tradeoffs and produce cost estimates.

A. MACRO APPROACH

The macro approach is concerned with man-loading across the life cycle of the project and, in particular, the maintenance phase. The basis for this approach is derived from the relationships pioneered by Norden and further developed by Putnam. As was stated in chapter two, the various phases of the software project life cycle have been found, in general, to be characterized by the Rayleigh curve function. The function is written as follows:

$$Y'_t = 2Kate^{-at^2}, \qquad (3.1)$$

where

Y' = manloading at any time t, normally measured in manyears or manmonths,

t = elapsed time from the start of the project,

k = the total accumulative manpower utilized over the project life cycle, measured in manyears or manmonths,

and

a = the shape parameter of the curve.

Norden demonstrated that the shape parameter (coefficient), a, can be calculated by the equation:

$$a = \frac{1}{2t_d^2}, \tag{3.2}$$

where

t_d = the point in time of maximum manpower utilization for the project.

It must be noted here that t_d in equation (3.2) can, in large projects (defined by Putnam as those projects with about 75,000 source lines of code [28]), be equated to project development time. In other words, large projects have historically been characterized by maximum manloading at the end of the development phase, roughly when the product was delivered to the user. However, it has been found empirically [29] that for other than large projects (less than 75,000 source lines of code) t_d actually falls at some point between t_0 and the end of the development phase. This may or may not affect the Green/Selby model. The end of the development phase will be denoted as t_{dev}, if it in fact does not coincide with t_d. Putnam has indicated that for small projects (less than 18,000 source lines of code) Y'_{max} is reached at about $t_{dev}/\sqrt{6}$. Medium sized projects (18,000 - 75,000 source lines of code) reach Y'_{max} somewhere between $t_{dev}/\sqrt{6}$ and $t_{dev}/2$. [30] Therefore, t_d, in this thesis, will be defined as the time at which Y' reaches a maximum.

Substituting equation (3.2) into equation (3.1) gives the following equation:

$$Y' = K/t_d^2\, te^{-t^2/2t_d^2} . \tag{3.3}$$

This equation can be used to calculate Y' at any point on the curve once K and t_d are known. The calculation or estimation of K and t_d have been sufficiently dealt with in the literature and so they will not be addressed here [31]. However, it must be noted that $t = t_d$ at the point of maximum manloading, and so, at that point, equation (3.3) breaks down to:

$$Y'_{max} = K/t\ e.^{-1/2} \qquad (3.4)$$

Norden also stated that the Rayleigh curve exhibited an inflection point where the decrease in manpower usage slows down in the descending portion of the curve [32], as characterized by the equation:

$$t_{ip} = (3/2a)^{1/2}, \qquad (3.5)$$

where

t_{ip} = the time of the inflection point of the Rayleigh curve, and

a = the curve shape parameter

The Green/Selby model is based in the theory that Y'_{tip} can be defined as a maximum level of maintenance effort for a project. The minimum level of maintenance effort is defined by Y'_{tim}, the inflection point on the curve for the maintenance phase, which, for large projects in general, has been said in the literature to follow the Rayleigh pattern. The definition of t_{ip} as a maximum level of maintenance was further supported by the hypothesis that the maximum level of manloading during the maintenance phase, Y'_{tm}, was equal to the manloading at the inflection point Y'_{tip}. This hypothesis appears to be based on the assumption that the maximum point of the maintenance phase coincides both in time and in magnitude with the inflection point of the life cycle curve. Green and Selby used the empirical data synthesized from a spectrum of USACSC projects to develop the theory. Figure (3.1) depicts their theoretical model.

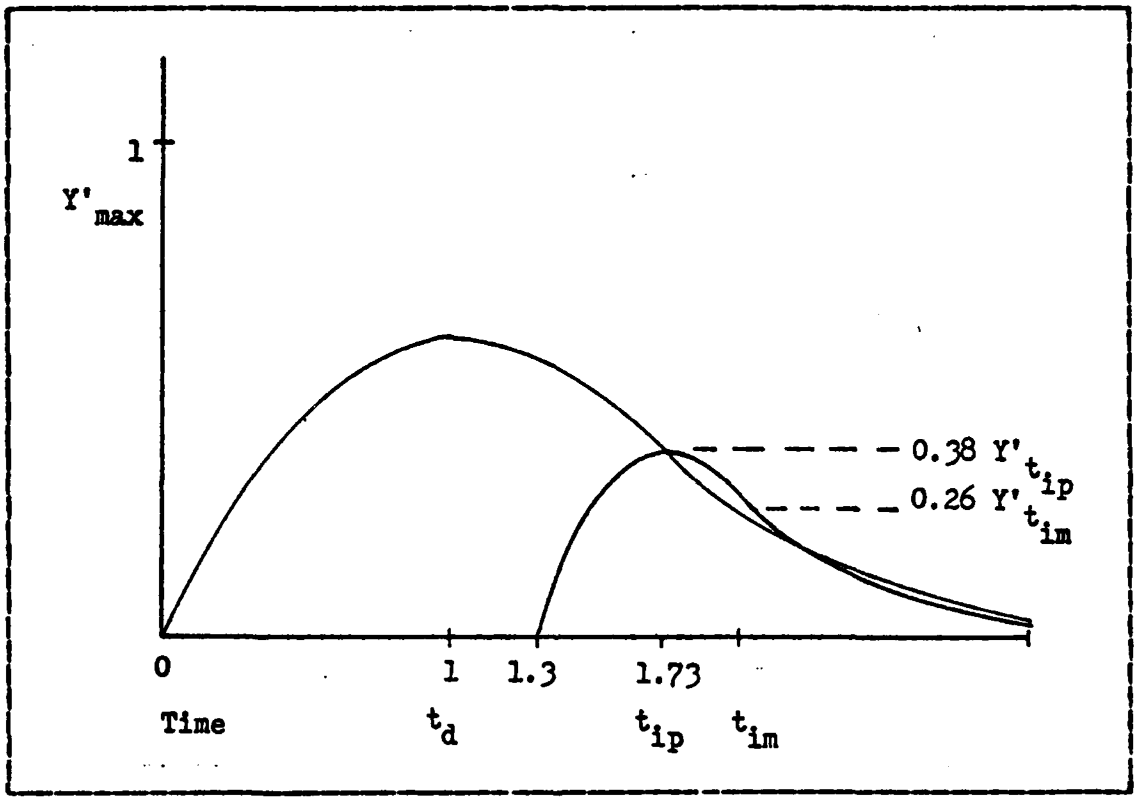


Figure 3.1 Normalized Rayleigh Curve

B. MICRO APPROACH

The micro approach was developed by Green and Selby using raw manning data obtained from the IBM Federal Systems Space Shuttle Program and the unpublished papers of Mr. Kyle Rone of IBM. This approach uses a matrix technique coupled with work breakdown structures to project maintenance manning requirements. The raw data was synthesized by Green and Selby to fit the macro model and then compared with the results of the micro matrix method. The authors of this thesis were not able to obtain data of sufficient complexity and refinement to apply micro techniques to it, and, therefore, the micro approach will not be discussed further in this work.

C. PROJECTED MODEL APPLICATIONS

The Green/Selby model was presented as a management tool. The control concept coupled with the planning concept appeared to be a total maintenance strategy package for the project manager. The model could provide management with the determination of a maintenance support level by use of the inflection point predictors (Y'_{tip} and Y'_{tim}) to define maximum and minimum maintenance manpower utilization boundaries. These boundaries, coupled with a planning strategy, provide a powerful planning tool.

Use of the model was also projected for forecasting of resource distribution via integration techniques applied to the area of the curve under the maintenance support boundary to break out manpower required by separation of development work (enhancements, additions, new design) from pure maintenance work (debugging, design error correction).[33]

The model was finally projected as a device for monitoring configuration control. Drawing on the work of Lehman and Belady, Green and Selby theorized that, as a project moves from pure "fix-it" type maintenance to modifications which may eventually lead to a new release of the product, the complexity of the product increases. This rise in complexity increases the maintenance level. As successive releases are developed, the maintenance level increases until it eventually exceeds the original maximum maintenance support level of the product. This would then predicate management assessment of the viability of the project from a cost effectiveness point of view, as the project will have reached what Green and Selby called a maintenance budget saturation point. At this point, or earlier, depending on management policies and desires, the old project would be terminated and a new life cycle/Rayleigh curve started.

D. CHAPTER THREE SUMMARY

The Green/Selby model appears to provide an easy-to-use cost estimation tool for the data systems manager. The macro and micro approaches give fairly quick estimates of maintenance manloading which can be cross compared and coupled with management constraints to fill out the system manager's overall strategy. If valid, it seems to partially fill the void in data systems management, alluded to in the GAO report, that of the lack of a maintenance strategy in an organization where maintenance is considered a discrete function.

IV. MODEL VALIDATION

A mathematical development of the Green/Selby model was completed by the authors of this thesis solely by algebraic substitution and reduction, working with the basic equations and relationships from the works of Norden and Putnam. An empirical development of the model was completed using the same or similar data to that used by Green and Selby. Both developments follow.

A. MATHEMATICAL DEVELOPMENT

The Norden/Rayleigh curve equation, as discussed earlier, is:

$$Y' = 2Kate^{-at^2} . \qquad (4.1)$$

This equation is characterized as a two parameter equation, as the outcome hinges on two parameters, K and a, calculated across the life cycle for all/any times from t_0 to t_n.

The parameter, a, as used in the Green/Selby model, is calculated by:

$$a = 1/2t_d^2 . \qquad (4.2)$$

The Green/Selby Model appears to have been developed for large projects with the assumption that t_{dev} and t_d do coincide. Therefore, if a is substituted into the Norden/Rayleigh equation, the commonly used form is found:

$$Y' = 2K*1/2t_d^2*t*e^{-1/2t_d^2*t^2},$$

which reduces to

$$Y' = K/t_d^2*t*e^{-t^2/2t_d^2}. \qquad (4.3)$$

Norden noticed that the inflection point on the project life cycle curve is characterized by:

$$t_{ip} = (3/2a)^{1/2}. \qquad (4.4)$$

If the equation for a is substituted in equation (4.4), t_{ip} reduces to:

$$t_{ip} = (3/2/2t_d^2)^{1/2} = (3t_d^2)^{1/2}. \qquad (4.5)$$

Substituting this equation into equation (4.3) gives:

$$Y'_{t_{ip}} = 2K(1/2t_d^2)\,t_{ip}\,e^{-((1/2t_d^2)(t_{ip}^2))},$$

which reduces to

$$Y'_{t_{ip}} = 2K(1/2t_d^2)(3t_d^2)^{1/2}\,e^{-((1/2t_d^2)(3t_d^2))},$$

which further reduces to

$$Y'_{t_{ip}} = 1.73K/t_d * e^{-3/2} . \qquad (4.6)$$

In the Green/Selby Model, it is theorized that the inflection point of the life cycle curve and the point of Y'_{max} on the curve for the maintenance phase coincide. The times t_{ip} and t_m are the same absolute time; however, for purposes of calculations, they differ, since t_m, the maximum manning for the maintenance curve is calculated relative to the start time for maintenance or the t_0 for the maintenance curve. If development time is equal to t_d, as was assumed in the Green/Selby Model, and if the maintenance effort starts at t_d, then the t_0 for the maintenance curve is t_d for the life cycle curve. Figure (4.1), with a corresponding time line, demonstrates the general relationship.

Green and Selby symbolized the elapsed time t_0 to t_m as t_e:

$$t_e = t_m - t_0 . \qquad (4.7)$$

It is at this juncture that difficulty in the development arises. The difficulty lies in the definition of where the maintenance phase begins. Does it begin at t_{dev} when the development phase ends as in Figure (4.1), or does it begin sometime after that? The time to Y'_{max} and thus, the shape parameter, a, depend on that definition. Green and Selby, using Army Data, stated that, on the average, the maintenance phase began at time 1.3 with t_d normalized to 1 or time (t_d + 0.3t_d). Therefore, the estimate of t_d for maintenance curve projection, or t_e, will be as shown in Figure (4.2) and equation (4.8) below.

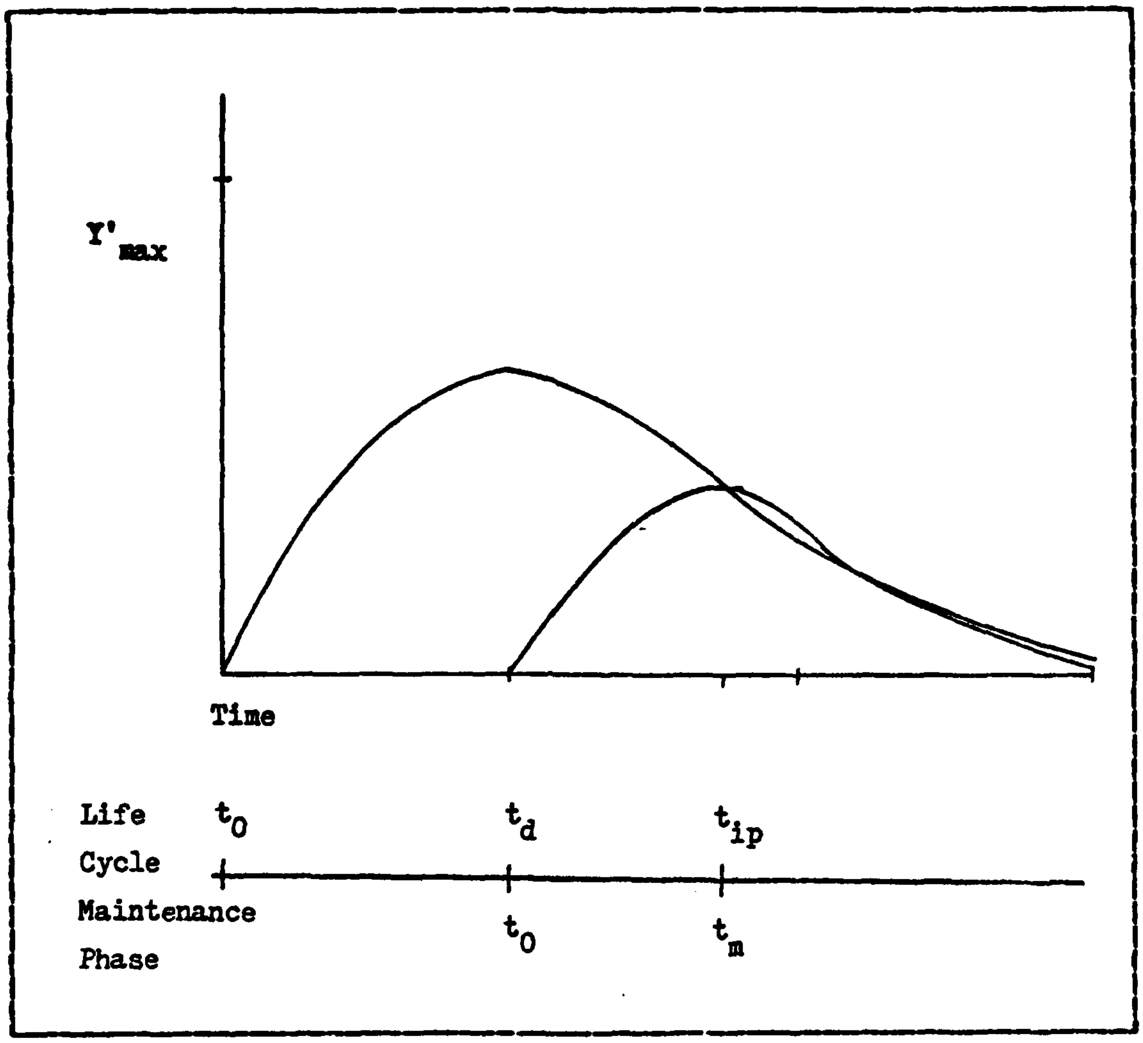


Figure 4.1   Maintenance Phase Timing Relationships

The estimate of K for the maintenance phase also came from the Army data which indicated that, on the average, the K for the maintenance phase is 20 percent of lifecycle K or 0.2 K (lifecycle) with lifecycle K normalized to 1.

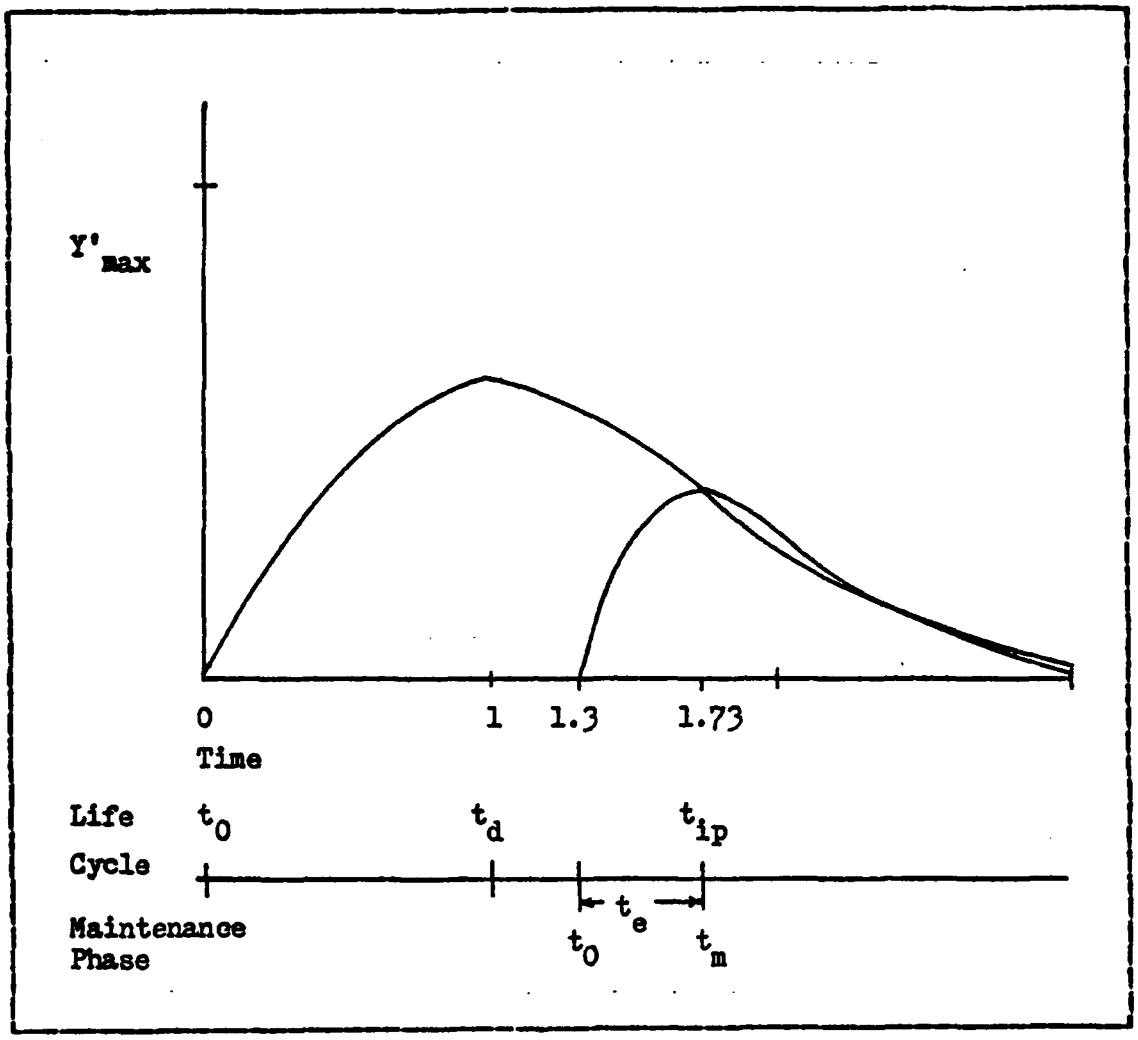


Figure 4.2 Maintenance Phase Timing Relationships in the Green/Selby Model

Since it is theorized that $t_m = t_{ip}$, it can be seen from Figure (4.2) that

$$t_e = t_{ip} - (t_d + 0.3t_d). \tag{4.8}$$

It must be noted here that this development, because of the nature of the problem and the lack of firm data, cannot be a pure mathematical development; however, the attempt is made to approximate it as closely as possible. Even though the t_d, or time of Y'_{max}, in the equations for the life cycle and maintenance curves denote the same type relationship within their parent equation, the quantities are necessarily different. As far as the authors know, and it is projected that the case was the same for Green and Selby, no specific relationship between t_d(lc) and t_d(m) have been found empirically. Therefore, for this development to exhibit credibility, known estimation factors from the Army data must be introduced. This also tends to indicate that until some firm relationship between t_d's is found, general applicability will be lacking. The same applies for the K factor.

After substituting the value for t_{ip} from equation (4.5), equation (4.8) becomes:

$$t_e = (3t_d^2)^{1/2} - (t_d + 0.3t_d) = 0.43t_d. \quad (4.9)$$

Substituting the value for t_e (maintenance phase t_d) into equation (3.4) for the Y'_{max} of a curve gives:

$$Y'_{t_m} = K/t_e * e^{-1/2},$$

which reduces to

$$Y'_{t_m} = 0.2K/0.43t_d * e^{-1/2}. \quad (4.10)$$

The constant e(-3/2), in equation (4.6), is calculated to be 0.223, and the constant e(-1/2) above is calculated to be 0.607. They are substituted into equation (4.6) and (4.10) respectively to give:

$$Y'_{t_{ip}} = 1.73K/t_d * 0.223 \quad \text{or}$$

$$Y'_{t_{ip}} = 0.386K/t_d \tag{4.11}$$

and

$$Y'_{t_m} = 0.2K/0.43t_d*0.607 \quad \text{or}$$

$$Y'_{t_m} = 0.121K/0.43t_d. \tag{4.12}$$

Attempting to equate $Y'_{t_{ip}}$ to Y'_{t_m} produces:

$$\frac{0.386K}{t_d} = \frac{0.121K}{0.43t_d}. \tag{4.13}$$

Algebraic reduction carries the development to completion:

$$\frac{0.43t_d}{t_d} = \frac{0.121K}{0.386K} \quad \text{and}$$

$$0.43 = 0.121/0.386 \tag{4.14}$$

which gives

$$0.43 \neq 0.313.$$

A similar development using K's and t_d's alone without the relational factors taken from Army project experience gives similar results. This is significant since it indicates that, for large projects where life cycle $t_d = t_{dev}$, the manloading at the maximum point on the maintenance curve is not necessarily equal to the manloading at the inflection point on the life cycle curve. There are situations where, theoretically, with the right values for t_d, t_e, and the two K's, Y'_{tip} and Y'_{tm} will be equal, but it becomes apparent that no such general rule can be demonstrated. Therefore, the proof of applicability, as has been the case in all areas of software cost estimating research so far, falls back into the arena of empirical development. The empirical development used by Green/Selby follows in section B.

B. EMPIRICAL DEVELOPMENT

The present authors, in recreation of the Green/Selby model, developed it as follows.

All parameters were normalized to values of t_d and K equal to 1. With $t_d = 1$ and equation (4.2) calculate a:

$$a = 1/2t_d^2 = 0.5. \tag{4.15}$$

Substitute a into equation (4.4) and calculate t_{ip}:

$$t_{ip} = (3/2a)^{1/2} = 1.73 \text{ years.} \tag{4.16}$$

Substitute t_{ip} into equation (4.6) to calculate $Y'_{t_{ip}}$:

$$Y'_{t_{ip}} = 2Kat_{ip}e^{-a(t_{ip})^2}, \text{ and}$$

$$Y'_{1.73} = 2(1)(0.5)(1.73)e^{-0.5(1.73)^2}, \text{ and}$$

$$y'_{1.73} = 0.387 \text{ manyears.} \tag{4.17}$$

To equate maximum maintenance manloading to the life cycle inflection point, define the time of maximum maintenance as t_m. Thus,

$$Y'_{t_{ip}} = Y'_{t_m}. \tag{4.18}$$

U.S. Army Computer Systems Command project data indicated that, across the spectrum of Army software projects, the maintenance phase included about 20 percent of the life-cycle. Therefore, K for the maintenance phase is 0.2K with respect to the normalized life cycle K value of 1. Here, it must be assumed that Army data analysis is valid. However, it is the contention of the authors of this thesis that an average of all Army large scale software projects will give a good figure for k/t_d for their types of projects. Army data also indicated that the maintenance phase started at 1.3 years normalized time (t_0). If $Y'_{tip} = Y'_{tm}$ at t_{ip}, then, making the same assumption as Green/Selby, that $t_{ip} = t_m$, the time of maximum maintenance manloading, t_e, can be calculated by:

$$t_m - t_0 = t_e, \text{ and}$$

$$1.73 - 1.3 = 0.43 \text{ years.} \qquad (4.19)$$

Calculate a_m for the maintenance curve from equation (4.2):

$$a_m = 1/2t_e^2 = 2.71. \qquad (4.20)$$

Substitute a and t_e into equation (4.1) to calculate Y'_{t_m}:

$$Y'_{t_m} = 2Ka_m t_e e^{-(a_m(t_e^2))},$$

$$Y'_{t_m} = 2(0.2K)(2.71)(0.43)e^{-(2.71(0.43^2))}, \text{ and}$$

$$Y'_{t_m} = 0.2824. \qquad (4.21)$$

Use equation (4.4) to calculate t_{im}

$$t_{im} = (3/2a)^{1/2} = 0.74 \text{ years.} \qquad (4.22)$$

The maintenance curve inflection point, t_{im}, on a life cycle basis, normalizes to 2.04 years. Substitute t_{im} into equation (4.6) to calculate $Y'_{t_{im}}$:

$$Y'_{t_{im}} = 2Ka_m t_{im} e^{-(a_m(t_{im}))^2},$$

$$Y'_{t_{im}} = 2(0.2K)(2.71)(0.74)e^{-(2.71)(0.74^2)}, \text{ and}$$

$$Y'_{t_{im}} = 0.182. \qquad (4.23)$$

The normalized curve as developed above is depicted in Figure (4.3).

Here, Y'_{tip} is clearly not equal to Y'_{tm}, as was also found in the mathematical development, but rather, Y'_{tm} is about 25 percent less than Y'_{tip} in magnitude, when t_m and t_{ip} coincide.

C. CHAPTER FOUR SUMMARY

In both the mathematical development and the empirical development, maximum manloading for the maintenance phase and manloading at the inflection point of the life cycle curve were not found to be equal. However, the maintenance maximum was below the magnitude at the inflection point.

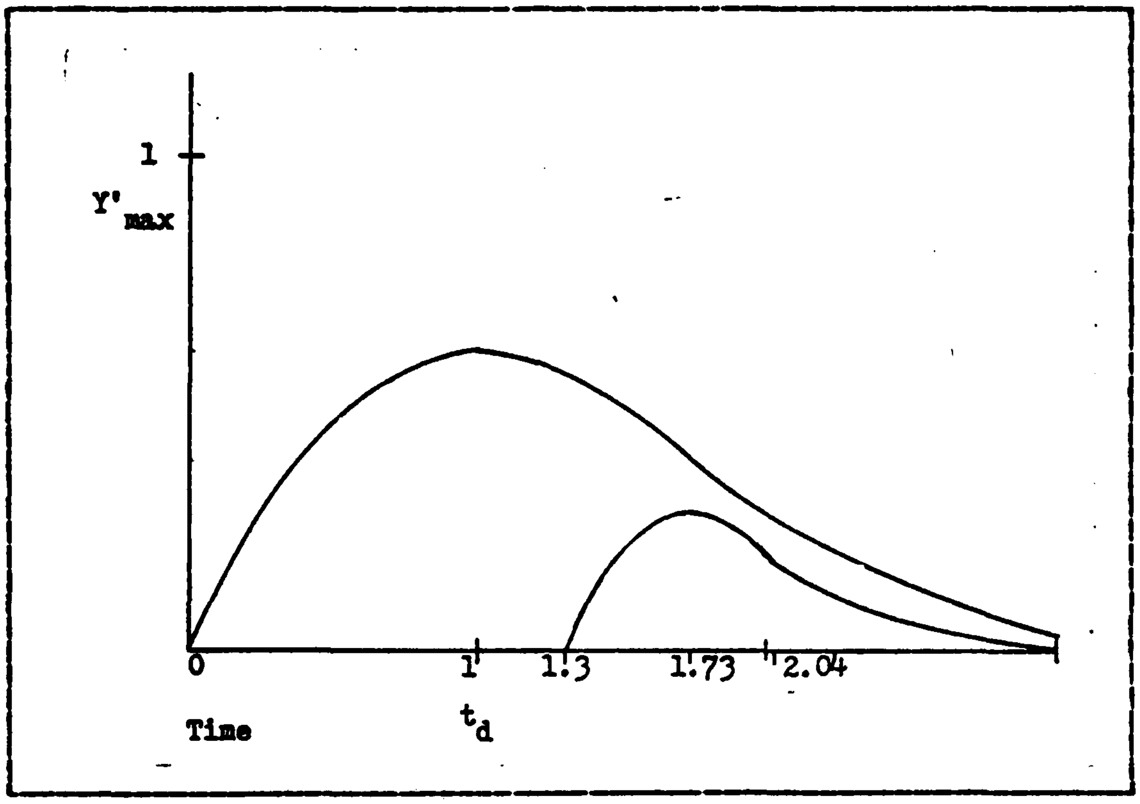


Figure 4.3 Developed Normalized Curve

Therefore, though the Green/Selby theory, in itself, may not be substantiated, some relationship/s may exist which can be used for maintenance manpower estimates. The key relationships in any maintenance manloading estimates appear to be those of life cycle K versus maintenance K and life cycle t_d versus maintenance t_d. If some empirical relationship (such as, for all large projects maintenance t_d is X percent of life cycle t_d or maintenance K is X percent of life cycle K) can be determined, then a model development could possibly be completed which produces fairly accurate manloading estimates. Such a model would not necessarily hinge on Y'_{tip} = Y'_{tm} but rather some relationship such as that exhibited by overall Army project data where Y'_{tm} or maximum average maintenance level fell at about 75 percent of Y'_{tip}. The difficulties encountered in attempting to develop the theory mathematically, in respect to ifferences in K's and t_d's, suggest that there may be other factors affecting the relationships and the parameters that determine those relationships. Such factors are discussed in Chapter VI.

V. RESEARCH ANALYSIS

A. DATA DEFINITION

The data utilized in the research effort was received from two sources, NASA Goddard Space Flight Center, Greenbelt, Md., and Dr. Willa Ehrlich, Bankers Trust Co., NY., NY. Both sets of data consisted of manloading for software projects over the life cycle and included maintenance data. Manpower utilization figures were in manhours for the NASA data and manmonths/mth for the Bankers Trust data. The NASA data was converted to manmonths/mth prior to analysis. The projects analyzed will be called NASA project and Projects A-D for the purposes of this thesis.

1. Bankers Trust Co. Data

Projects A-D were all medium sized projects, developed at Bankers Trust Company. The few project characteristics that were known can be found in Table V. A listing of project data by manmonths/mth is found in Appendix C.

2. NASA data

NASA project data were related to an operational system and, though it is an ongoing project and the complete life cycle is not yet known, much information could be synthesized from the life cycle and maintenance data to date. Pertinent project characteristics are listed in Table VI. It is readily apparent that the project started as a small project, but that it has migrated via maintenance to what could be called a large project. However, based on project size at the end of development, it must be classified as a small sized project. A listing of project data by manmonths/mth is found in Appendix C.

TABLE V

Bankers Trust Co. Projects Characteristics

Project Name	Size	Development	Maintenance	Ending
A	Medium	8/78	1/80	12/80
B	Medium	8/79	6/80	4/81
C	Medium	12/76	4/78	12/78
D	Medium	3/77	11/77	12/79

B. ANALYSIS PROCESS

The analysis process fell into two categories, curve fitting, and comparison. Actual life cycle manmonth figures for individual projects were fitted against the Rayleigh equation via the facilities provided for non-linear curve fitting in the Statistical Analysis System (SAS) package available on the resident IBM 3033AP Computer System. The Marquardt method was chosen as the regression technique. In addition, data from the four Bankers Trust Co. projects were combined by normalizing t_d (the time to reach Y'_{max}) to 1 for each project and then the curve fitting techniques were applied to the normalized/combined data. Manpower figures for the maintenance phases of individual projects and the combined data were also fitted to the Rayleigh equation and then, in each situation, actual data points and fitted curves for life cycle and maintenance phases were replotted on a common axis to provide an aggregate picture of the phase relationships.

The USACSC data was also reanalyzed. Though it did not provide substantiation for the specific theory of Green and Selby, as noted in chapter four, it does provide valuable

TABLE VI

NASA Project Data Characteristics

PROJECT HISTORY		
A.	Design start date	March 1, 1975
B.	Maintenance start date	July 30, 1977
C.	Date of last data	January 25, 1982

CODE HISTORY	
A. Lines of Code	
1. Original lines of code	4,000
2. Modified lines of code	8,141
3. New lines of code	61,230
4. Total lines of code	73,371
B. Modules	
1. Original modules	35
2. Modified modules	75
3. New modules	450
4. Total modules	560
C. Documentation	
Pages	3,300

insight into the phase relationships as applied to large sized projects. A mass of raw data was not available, but by taking the aggregate figures provided, critical points along the Rayleigh curve were calculated.

After the curve fitting was completed, the parameters K, a, and t_d for the life cycle curves and the corresponding maintenance curves were compared to examine possible common relationships. Curve magnitudes at t_{ip} for the life cycle and $Y'_{max}(t_d)$ for the maintenance curve were also compared in terms of the general relationships proposed by Green and Selby.

C. ANALYSIS RESULTS

An excellent fit was obtained for the life cycle curves for all five individual projects in relation to the Rayleigh model. From Table VII, correlation coefficients ranged from $r^2 = 0.776$, for the NASA project, to $r^2 = 0.966$, for Project A. The curve fit for the combined Bankers Trust projects obtained an $r^2 = 0.869$. However, maintenance curves, in general, did not fit the Rayleigh model well, with correlation coefficients ranging from $r^2 = 0.118$ for NASA data to $r^2 = 0.762$ for Project B. Projects B and D maintenance curves best fit the Rayleigh model with $r^2 = 0.762$ and 0.747 respectively. These findings indicate that the maintenance efforts are somewhat erratic, as alluded to in the GAO study, and, therefore, do not fit a specific curve well. When maintenance is not managed as a discrete function, manloading peaks and drops in an inconsistent manner. This normally results as managers respond, on a crisis basis, to provide maintenance activity only when trouble arises.

In the NASA data, however, though the overall maintenance data does not fit the Rayleigh curve well, visual inspection of the curve reveals what appear to be a series of small Rayleigh-like curves, the combination of which exhibit an overall rise of maintenance manloading across the available data, as can be seen in Figure (5.1).
This trend fits well with the project characteristics which show that the size of the project has grown from 4000 SLOC to about 73,000 SLOC during its life cycle to date. It

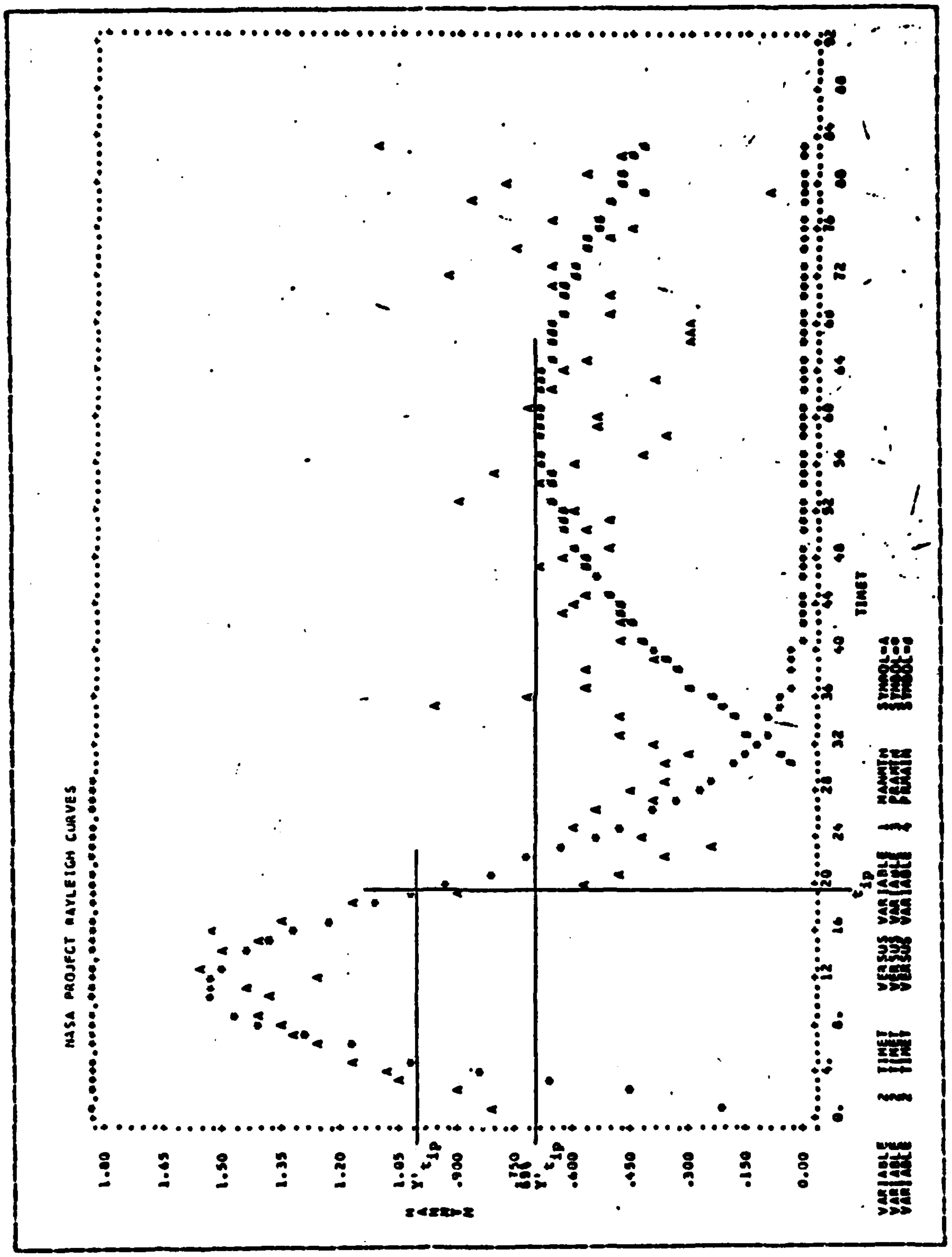


Figure 5.1 NASA Data Fitted Curves

stands to reason that the "mini-development cycles" for those modifications/enhancements which created the increase in system size would, themselves, exhibit a Rayleigh pattern, but the aggregate maintenance phase would not necessarily follow the same pattern. The aggregate curves are included in Appendix C.

Comparison of parameters gave varying results, as can be seen in Table VII. Ratios of life cycle K's to maintenance K's ranged from 0.148 to 1.24 and ratios of life cycle t_d's and maintenance t_d's ranged from 0.625 to 2.82. This seems to indicate that no general relationship can be derived which relates K's and t_d's for the maintenance phase versus the life cycle with respect to individual projects. However, as more data is accumulated and research efforts continue, those relationships might be found to exist for various aggregate projects.

When Y'_{tip} of the individual fitted life cycle curves was compared to Y'_{tm} of the individual fitted maintenance curves, similar results to those obtained for K and t_d comparisons were observed. The ratios covered a wide spectrum. However, when the comparison was made for the combined Bankers Trust projects curves, the results were strikingly similar to those of the NASA project and the USACSC data. USACSC data indicated, as shown in Chapter IV, that, on the average, Y'_{tm} = 75 percent of y'_{tip}. Comparison of actual maximum manloading for the combined Bankers Trust project data to the inflection point on the fitted life cycle curve gave Y'_{tm} = 69.6 percent of Y'_{tip}. Though only one project, instead of an aggregate, the NASA data also showed a general behavior of Y'_{tm} = 69 percent of Y'_{tip}. For the NASA project, this interpretation may be questionable, since some data points lay above the 69 percent of Y'_{tip} level. In fact, one point lay above Y'_{tip}.

TABLE VII

Compilation of Analysis Results

Life Cycle Parameters

NAME	a	t_d	K	Y'_{max}	y'_{tip}	tip
NASA Project	.003969	11	28.410	1.54	0.9982	19.44
Project A	.007143	8	183.374	13.27	8.8586	14.49
Project B	.014294	6	137.276	14.08	8.8422	10.25
Project C	.007605	8	186.913	13.98	9.0296	14.04
Project D	.024288	5	81.383	10.77	6.2905	7.86
Comb'n A-D (norm. td=1)	.598560	1	19.435	12.89	8.2190	1.58

Maintenance Phase Parameters

NAME	a	t_e	K	Y'_{max}
NASA Project	.000525	31	35.234	0.693
Project A	.022420	5	27.165	3.477
Project B	.019000	5	47.204	5.579
Project C	.006000	7	53.127	4.000
Project D	.005900	9	56.699	3.740
Comb'n A-D (norm. td=1)	.311000	1.26	8.480	4.080

Miscellaneous Parameters

NAME	$\frac{td(M)}{td(LC)}$	$\frac{K(M)}{K(LC)}$	$\frac{Y'tm}{Y'tip}$	Main Corr.	Life Cycle Corr.
NASA Project	2.820	1.24	.694	.118	.776
Project A	0.625	.148	.392	.511	.966
Project B	0.833	.343	.631	.762	.872
Project C	0.875	.284	.443	.482	.939
Project D	1.800	.696	.595	.747	.893
Comb'n A-D (norm. td=1)	1.260	.436	.496	.388	.869

However, if one accepts the theory that the NASA project is characterized during the maintenance phase by a series of "mini-development phases", then the points above the 69 percent level can be interpreted as manning levels intrinsic to the development effort and not characteristic of a general maintenance program. Then the aggregated maximum maintenance level lies at 69 percent of Y'_{tip}.

D. CHAPTER FIVE SUMMARY

The data were analyzed using non-linear curve fitting techniques to provide life cycle versus maintenace phase relationship comparisons. The results seem to exhibit independence of behavior with respect to values of K and t_d. However, a general trend, within the limited scope of data available, was found which appears to point to a possible relationship between maintenance manloading levels and the magnitude of the inflection point on the life cycle curve.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. INTRODUCTION

The history of the software industry has been marked by cost overruns, late deliveries, poor reliability and maintenance, and user dissatisfaction. While these problems are not unique to computing, the record seems to indicate that software developers as a group are less successful in meeting quality, cost, and schedule objectives than their hardware counterparts.[34] With this in mind, a number of models have been developed, as discussed in Chapter II, to provide management the necessary tools to more accurately predict the actual costs and time frames for their software projects. This thesis attempted to expand the work done by Green and Selby on Putnam's model, with special emphasis on the maintenance phase of the software life cycle. This included a detailed comparison of the peak manloading for the maintenance phase with the inflection point on the total life cycle Rayleigh curve.

B. CONCLUSIONS

The software project manpower macro-estimating model, as presented by Green and Selby, is not a usable model for the project manager. As was demonstrated in Chapter IV, and again in the data analysis in Chapter V, the maximum point on the maintenance curve is not necessarily equal to the magnitude at the inflection point of the life cycle curve, though, theoretically, it is possible for the two points to be equal. It was also found that the absolute point in time of the maximum maintenance manloading and the inflection point may coincide, but, usually, will not. However, these findings do not invalidate the basic ideas from which the Green/Selby model were developed. Those basic ideas were that a relationship may exist whereby maintenance manpower could be projected by comparison of the maintenance phase and life cycle Rayleigh curves, or derivations thereof. It was shown that, within the scope of the limited available data, only two of the five projects analyzed were characterized by maintenance phases which closely fit the Rayleigh model. However, it was demonstrated that, for combined project data, within project type, and within a specific organization, a relationship does appear to exist between the maximum maintenance manpower utilization level and the inflection point of the life cycle curve, whether the maintenance phase fits the Rayleigh model or not.

In both the USACSC and combined Bankers Trust Co. data analyses, and with interpretive license in the NASA data analysis, maximum maintenance levels were within 65 percent to 75 percent of the level of Y'_{tip}. There is not enough evidence here to show that there exists a general rule that maximum maintenance will be about 70 percent of the magnitude at the life cycle curve inflection point, but the implications for project managers within individual organizations are encouraging. The results of the data analysis appear to indicate that, for project type, within an individual organization, analysis of historical data and comparison of maintenance levels to life cycle curve inflection points will provide a general baseline maximum maintenance support level which the manager can use in outyear maintenance manning projections for future projects. For example, if historical data for accounting type projects in organization X shows that maximum maintenance manning is 65 percent of the magnitude at the life cycle curve inflection point, then the manager can apply that percentage to the projected life cycle curve calculations for future projects to obtain a maintenance support projection at the inception of the project. As the life cycle curve is refined during the development phase, the maintenance level projections can be successively refined. This would provide the ADP manager with a valuable tool in an environment presently characterized by a general lack of planning and management direction, in the area of software maintenance.

The results of the data analysis further indicate, by their lack of strong correlation, that there are other factors which may have a strong effect on the level of maintenance required for any software system. This finding is not entirely surprising, as the authors of this thesis, after extensive readings in the literature, did not have much confidence in the possibility of discovering a single, general, simple decision rule for software maintenance manning. Rather, the research completed here is only a tiny bite taken from the mountain of research which needs to be done. The possible set of constraints and combinations thereof which affect the software process is astounding. A few were highlighted by this research effort. It was found that there was no firm relationship between K's and t_d's of the corresponding life cycle and maintenance phase curves. It can be hypothesized that differences in K's (total life cycle manning) are attributed to such factors as project size, complexity, and project type. It follows that larger projects will require higher overall manning levels than smaller sized projects. The relationships of maintenance t_d versus life cycle t_d are affected, in large part, by complexity and size of the project. Differing system complexities may place heavier burdens on different phases of the development processes, and, thus, cause t_d (time of maximum manning) to occur at different times for different projects. There may be, and the authors of this thesis feel that there will be, no definable relationship between the point of maximum manning for the maintenance phase and the corresponding td for the life cycle. Since only two of the five projects analyzed actually fit the Rayleigh model for the maintenance phase, it would appear that for some projects, a definable t_d would be forever elusive. Only in those projects where some type of "mini-development" effort is completed in the process of providing enhancements or major modifications will a good fit to the Rayleigh model be realized, accompanied by a definable maintenance t_d versus life cycle t_d relationship for that project.

A constraint of even greater importance is the use of varying software development techniques and methodologies. It has been speculated that the majority of research to date has been conducted with data collected from projects which were characterized by design and coding efforts which did not include structured, modular-design techniques, information-hiding modules, and other software development concepts and tools. These projects have shown a very close relationship with the Rayleigh model. A tremendous impact on the entire arena may be seen with the increased use of the above listed design techniques. How these techniques will affect the software equation and, in particular, software maintenance, is yet to be seen.

The rise in maintenance activity for the NASA project, as new developments apparently added modules and source lines of codo to the system, seems to support the results obtained by Lehman and Belady, as described in Chapter II, that, as enhancements are added to the original project, the maintenance level required to support the project also rises. This could be attributed to the fact that the addition of enhancements adds complexity to the system which, in turn, causes a resultant increase in the maintenance level required. As was discussed earlier, and as is seen in the NASA project data, if enhancements continue, the maintenance manning rises above the magnitude of the inflection point on the life cycle curve. This could also indicate that the point in time at which the project should be totally rewritten and restructured as a new project has been reached, and any further development-like effort on the system should constitute the inception of a new project.

C. RECOMMENDATIONS

One of the most difficult problems encountered in the preparation of this thesis was locating organizations which had compiled and/or retained historical data from their software development and maintenance efforts. Some of the organizations contacted had maintained some form of historical data, but they had not broken their information down into a format which could be used to obtain information about the separate phases of the software life cycle. Therefore, if any meaningful research is to be conducted in the future in this area, organizations which are responsible for producing or maintaining software products need to start accounting properly for the various costs associated with this process. Proper accounting includes, not only tracking the number of source lines of code produced for the project, but total man-hours expended in each phase, the actual time frame for each phase, and the applicable complexity factors. The collection of this data, however, must be an ongoing process, just as is proper documentation of software, and it should become a part of this documentation. By making the collection process an ongoing process, the data is always current, and less subject to error. For, like any other form of documentation, if postponed until the end of the project, it is subject to a host of errors, omissions, and inaccuracies. However, even if the collection process is done with total perfection, it means nothing unless the data is recorded in such a manner that it can be retrieved and understood easily. It is therefore recommended that this data be stored in an automated data file so that it can be accessed quickly and analyzed with greater ease and efficiency than with a manual system. With the cost of software rising at an ever increasing rate, the benefits of this information to the organization, seem obvious. Not only should it be better able to predict future software manning requirements, but also, it should be able to identify and correct other inefficiencies within the development and maintenance processes.

As noted by GAO, and as indicated by the NASA data, a generally accepted but uniform definition of software maintenance is not now in existence in the majority of organizations. In addition, management is not presently requiring that software maintenace be managed as a discrete function. This leads to many problems for management at various levels of the organization. As such, it is recommended that the definition proposed by GAO be adopted as the uniform definition of software maintenance. It also is recommended that software maintenance be accomplished as a discrete function within the organization. The adoption of the GAO definition will leave a grey area where enhancements to the old project stop and a new project begins. However, if management formulates a project maintenance strategy which includes the development of a maintenance support level, whether it is based on a percentage of the magnitude at the inflection point on the life cycle curve, or on some other management-defined function, a point will exist above which management should decide to terminate enhancements to that project and start a new project. This project would be developed as a follow-on to the old system. The old project should be terminated or continued with a minimum maintenance support level to effect necessary repairs until the new system comes online.

Although there appears to be a strong correlation between peak maintenance manloading and a fixed percentage of the manloading at the inflection point of the total life cycle Rayleigh curve, further work needs to be done to determine if this relationship holds true throughout the software industry. This work should include comparisons across all types of software and comparisons within each class to determine if there is a value that management could use as a planning tool for the type of software they are producing. Follow-on research to this thesis would be most beneficial if completed in the following manner. A larger base of life cycle/maintenance data must be collected to provide a better picture of the relationships concerned and to obtain a higher percentage of validity in the findings. Projects need to be analyzed individually, grouped by project size, grouped by type of system involved, grouped by complexity factors (if known), and grouped within specific organizations as well as a total combination of the collected population. Research should be done to examine potential relationships of K's, t_d's, and Y'_{tip} versus Y'_{tm} for the corresponding life cycle and maintenance curves. A particularly important area of research will be the effect of new software development techniques on the software equation. Any data collected on projects which were developed in this manner should be segregated and analyzed separately. The potential for research in this area is unlimited in scope and in promise.

APPENDIX A

ANALYSIS OF SOFTWARE MODELS BY THIBODEAU

A. INTRODUCTION

Robert Thibodeau, while working for General Research Corporaton, was contracted by the Air Force to conduct a study of the various models currently avalilable for software cost estimation. This appendix consists of excerpts from his review.

B. AEROSPACE MODEL

Description of the Model

The model was developed using regression techniques applied to data from software development projects characterized by one-of-a kind computers, limited support software, software, special languages and severe memory size and speed requirements. The data were stratified into two groups. One group contained 13 projects for the development of real time software identified as primarily large-scale airborne and space applications. The second group consisted of 7 operational support programs presumably without the size and speed requirements of the first group.

The model description is not clear concerning the exact composition of the estimate of effort required to develop the software. Only the total effort is extimated. The estimate is made using a relationship of the form:

$$MM = a\ (\text{Instruction})^{b}$$

where the constants, a and b, are determined by regression analysis.

The estimating relationships are:

Real Time Software

$$MM = 0.057\ (I)^{0.94}$$

Support Software

$$MM = 2.012\ (I)^{0.404}$$

where:

MM = total development effort, manmonths

I = number of instructions (independent of language)....

C. DOD MICRO ESTIMATING PROCEDURE

Description of the Model

The primary estimating relationship comprising the DoD Micro Procedure can be described as the ratio of a factor representing the software to be developed or changed and a productivity measure.

The model form suggests that effort increases directly with the number of input and output configurations operating on the system being built. Effort also increases with the number of routines being created or modified weighted by their difficulty. The total effort is scaled according to the amount of work that must be done in entirety as opposed to modification of an existing system.

The number of days needed to deliver the product (effectively the days of effort per unit of product) depends on the general experience and accomplishment of the development group (measured by their job classifications) weighted by their knowledge of the problem to be solved relative to the knowledge required. One other factor that directly affects the productivity is the ease of access to the computer (measured by turnaround time).

the basic form of the estimating relation for software development time is:

Net Development Time = (Product) / (Productivity)

Where:

Product is a measure describing the effort to be performed.

Productivity is the rate of creating the product from the application of personnel time.

Product = (Number of Formats + Weighted Number of Functions) x (Effort Relative to a New Development)

The terms in parentheses along with the following terms are defined in the discussion of model inputs below:

$(\text{Productivity})^{-1}$ = (Work Days per Unit of Product for a Staff with Average Experience)
x (Job Knowledge Required)
x (Job Knowledge Available)
x (Access)

The result is the total hours required for code development. Presumably this means detailed design, coding, and unit testing.

Gross Development Time = (Net Development Time)
x (Other System Factor)
x (Non-Project Factor + Lost Time Factor)

A value of 1.8 is recommended for the other system factor. This factor represents the effort needed to convert the code development time to total development time. This value is representative of an observed range from 1.2 to 2.1. Total development includes analysis, design, coding, testing and documentation. It is the sum of the project direct charges. Whether this includes support hours for clerical and other functions is not clear. but any given organization could include these by modifying the 1.8 factor.

The net development time accounts for the time lost from normal scheduled working hours for leave, sickness, holidays, and non-project assignments. These add 25 percent to the total development time. There is also a 10 percent efficiency factor (coffee breaks, time cards, code rework, etc.). The code rework should probably be handled elsewhere. It is probably included where it is to make the 10 percent palatable. It should be included in the gross size adjustment and the 1.8 factor.

The effect of these adjustments is to estimate the number of personnel who must be assigned to the project to ensure delivery of the total development hours. These factors are orgainizational specific.

Although the resource estimating procedure includes weighting factors for the input and output formats by type of device (see subsequent discussion), the factors have a value of one in each case. Therefore, the model describes a linear relationship between the total number of file formats and the effort required to implement them. It may be that future versions of the model will weight the types of file device differently. Then the effort required to implement a report format may be different from the effort required for a card format.

Program complexity, which is the second term in the product measure, is the weighted sum of the functions to be implemented. The weights depend on the function and its assumed level of complexity. The weights range from 1 for a simple operating system control language change to 12 for a very complex edit-validation function.

The value 3 is the most common among the 24 possible function-complexity assignments. If the function types are equally represented in programs, the average value is 4.

The programmer/analyst experience factor is an indication of the effect of experience on productivity. Values range from .75 to 2.75 corresponding to a lead analyst to programmer and interns respectively. Since experience is not evenly distributed over a group of programmers and analysts, the following groups was hypothesized in order to obtain an average or representative value for the experience factor.

Experience	Number in Group	Factor	Weighted Sum
lead	1	.75	.75
Senior	2	1.25	2.50
Journeyman	4	1.75	7.00
Nominal	8	2.25	18.00
Intern	5	2.75	13.75
	20		42.00

Average Value = 42 / 20 = 2.1

No definitions are provided for the 10 job classifications. The job knowledge and turn-around time factors are self-explanatory.

The System Factor adjusts the product development effort to account for work already done. The product measure resulting from the format count and the program complexity value is the same whether the system is being developed in its entirety or it is a modification to an existing system. The system factor has the effect of modifying the product value to account for less than total development.

Seven levels of change are described by the System Factor. The values range from 2 for a new development to 8 for an operating systems control language change.

For a new system development the 2 in the primary estimating equation is divided by a System Factor value of 2 and the product measure is unchanged. Consequently, the System Factor values describing lesser amounts of new development have larger values and are portions of 2. The effect of the System Factor on the product measure is summarized as follows:

Type of Effort	System Factor	Effort Relative to a New Development
New Development	2	1.00
Major Change	3	.67
Major Modification	4	.50
Minor Modification	5	.40
Maintenance	6	.33
Minor Technical Change	7	.29
Operating Systems Control Language Change	8	.25

In order to get a feel for the relative magnitudes of the components of the Micro Estimating Procedure, consider the following example.

Number of I/O formats = 10
Number of functions = 20
Average complexity factor = 4.
New Development
Product = (Number of Formats + Weighted Number of Functions) x (Effort Related to a New Development)
Product = (10 + 4 x 20) x 2 / 2 = 90
Experience = 2. (See above for computation)
Job knowledge required = 1.0
Job knowledge available = 1.0
Access = = 1.0
(Productivity) = (Work Days per Unit of Product for a Staff with Average Experience)

x (Job Knowledge Required)

x (Job Knowledge Available)

x (Access)

= 2.0 x 1.0 x 1.0 x 1.0 = 2.0

Net Development Time = (Product) x $(\text{Productivity})^{-1}$

= 90 x 2.0 = 180 Man-Days

If the effort was a major modification (System Factor = 4), the Product value becomes:

product = (10 + 4 x 20) x 2/4 = 45

and

Net Development Time = 45 x 2.0 = 90 Man-Days

If the Job Knowledge Required is "Detailed" (Factor = 1.5) and the Job Knowledge Available is "Limited" (Factor = 1.5), and the productivity becomes:

$(\text{Productivity})^{-1}$ = 2.0 x 1.5 x 1.5 x 1.0 = 4.5

then for the major modification:

Net Development Effort = 45 x 4.5 = 202.5 Man-Days

<u>Outputs</u>

The primary output (i.e., the output that is sensitive or controlled by project variables as opposed to the subsequent step which is a fixed allocation) is: Gross Development Time (man-days). Gross Development Time includes:

- Nonproject time (individual assigned to project but busy with nonproject tasks, e.g., training, nonproduct administrative duties, etc., and vacation and holidays)
- Wasted or lost time

therefore, Gross Development Time describes the staffing level that will result in a needed amount of development time. The latter is predicted by program and project characteristics.

The secondary outputs (i.e., those derived by applying fixed values to the primary output are:

- Effort by project phase
- Total development cost

The project phases are:

- Review and analysis
- Design
- Programming
- Testing
- Documentation

Gross Development Time includes:

Analysis of present methods
Design of the new/changed system
Develop the system's support
Program design
Program development
Program testing
System testing
Installation and conversion
Staff training
Project officer
System manager
Technical managers
Support personnel
Documentation

Inputs

Product Related Inputs. The software is described by the numbers of types of items it processes and the numbers of functions it includes. The functions are described according to type and complexity. The result is two product descriptors: one measures the size of the input/output processing to be executed by the system; the other is a measure of the number and difficulty of the functions to be performed.

Input File Formats. The number of different formats to be read by the system are counted and added together. The model asks for numbers of card, tape, disk, and screen formats separately, but since the weighting factor is always one, there is no distinction made among them regarding the effort involved to implement them.

Output File Formats. The formats output by the system are totaled. The same entries as for the inputs are requested plus the number of report formats. As in the case of the inputs, the weighting factor for the different types of output is always one, so there is no reason to differentiate.

Program Complexity. The total program complexity measure is computed by a weighted sum of the number of processing functions of given types. Each function is characterized as simple, complex, or very complex. The processing functions are:

- Edit Validation
- Table Look-Up (Internal or External)
- Calculations
- Sort/Merge Process
- Internal Data Manipulation
- File Search
- Utilities or Subroutines
- Operating Systems Control Language

Job Knowledge Required. The amount of knowledge required to implement or change a system has a direct effect on the number of hours required to accomplish the project. A system that requires very detailed knowledge will require more effort than one that can be accomplished with limited knowledge. This parameter is paired with the job knowledge available factor described below to describe the relative influence on productivity. Three job knowledge levels are used: Limited, General, Detailed.

System Factor. The effort required to complete a system development or change project of given complexity depends on the state of the system. That is, the work required to develop a system with three file formats, all other factors being equal. The System Factor describes the level of effort being undertaken. Seven levels are described:

- System development
- Major changes
- Major modification
- Minor modification
- Maintenance
- Minor technical change
- Operating systems control language

Resource Related Inputs

Programmer/Analyst Experience Available. The available experience measure is an effective productivity indicator. It quantifies the rate at which the product can be produced in terms of the job classification of the staff available for assignment to the system development. Two data processing personnel classifications: Analyst and Programmer, are tabulated according to five levels of experience: Lead, Senior, Journeyman, Nominal, and intern. Weights are associated with the difference experience levels. The result is a weighted average productivity factor.

Job Knowledge Available. This factor has the effect of describing the change in productivity associated with the level of knowledge about the work to be performed that exists among the persons available for assignment. It works together with the Job Knowledge Required factor described above to quantify the effect of the knowledge of the system required compared to that available on the time required to complete the work. In general, the effect of the combined factors is to increase the development manhours if the need exceeds the available and decrease the hours if the available exceeds the need. Three levels of job knowledge availability are specified: Limited, General, and Detailed.

__Program Turn-Around Time__. The effect of computer access on productivity is described by four levels of average turn-around time:

- Interactive terminal
- More than one run per day
- One run per day
- Less than one run per day.

D. DOTY ASSOCIATES, INC.

__Description of the Model__

The model is actually a set of 15 estimating relationships. Each one to be used for a given type of software and software life cycle phase. Equations have been derived empirically using regression analysis for the following types of software:

- Command and Control
- Scientific
- Business
- Utility

The development effort for software representing each of the application types may be estimated using one of three different relationships. An additional three are given that are applicable to all types of software. These equations are to be used "when the application cannot be categorized or is different than the categories noted". The procedure specifies that when a software system is made up of subsystems that are different types, the total size should be divided into the four catagories and the appropriate estimating equation used for each one. Then the individual manmonths are summed to give a total system development effort. The three equations are divided into size measure (lines of source code or words of object instructions) and the life cycle phase in which the estimate is made (Concept Formulation and all others). If the estimate is to be made using the words of object instructions, the same equation is used in all life cycle phases. Similarly, for estimating large systems (more than 10,000 lines) using lines of source code requires the use of a different equation in the Concept Formulation Phase than in the other life cycle phases.

The use of the different equations can be described as follows (A, B, and C refer to the three different relationships).

SOFTWARE DESCRIPTION	LIFE CYCLE PHASE	
	CONCEPT	OTHERS
WORDS OF OBJECT CODE	A	A
LINES OF SOURCE CODE LARGE SYSTEM ≥ 10 K LINES	B	B
SMALL SYSTEM > 10 K LINES	B	C

The forms of the estimating relationships are similar. Equations A and B are of the form:

$$MM = a\ I^{b}$$

where

MM = Manmonths of development effort.

I = either words of object code (A) or lines of executable source code (B).

a,b = Constants obtained empirically.

Equation C has the form:

$$MM = cI^{d} \sum_{j=1}^{14} f_{j}$$

Where

f_j = a set of parameters describing the development environment.

c,d = constants obtained empirically....

The following guidelines are presented for selecting the proper estimating relationship.

- In Concept Formulation, if the size of the program in object code is known, use the object code estimators. They will give more accurate estimates of manpower requirements.
- If accurate estimates of manpower requirements are required in the Analysis and Design and subsequent phases of development, use equation B, in source code, for programs of $I \geq 10{,}000$ and equation C, in source code, for programs with $I < 10{,}000$.
- For budgetary purposes, use the equation that gives the higher estimate.

Development time is estimated using the equation

$$D = \frac{1000I}{92.25 + 233I^{.667}}$$

Where

D = Reasonable development time in months

I = number of delivered object instructions.

This relationship was obtained using regression on data describing 74 development projects. The time estimate should describe "customary" distributing of effort over time that is, it should avoid extremes of project time compression or expansion.

It should be noted that a large portion of the documentation accompanying the description of the DAI estimating procedures is devoted to discussions of factors that are believed to influence the cost of software development.

These factors are classified according to aspects of software and its development environment. The factors are grouped according to the following "domains":

- Requirements
- System Architecture/Engineering
- Management

<u>Outputs</u>

<u>Cost of Software Development</u>

The estimate of total development cost is based on several relationships that portion the cost into components that can be estimated by applying available ratios to other costs and factors such as overhead and administrative costs. By the proper use of relevant values for these factors the relationships can represent either goverment in-house costs or contractor development costs. A method is described for time phasing the expenditure that is said to satisfy the requirements of DoD Directive 5000.1.

The procedure identifies costs that are incurred by the government during all phases of the software life cycle <u>except</u> Operation and Support. The total development cost includes:

$$C = C_{CF} + C_{VAL} + C_{FSD}$$

where

C	= Development Cost
C_{CF}	= Conceptual Phase Cost
C_{VAL}	= Validation Phase Cost
C_{FSD}	= Full Scale Development Cost.

Information is included that relates the government cost to the contractor's full scale development cost. This cost is the one developed by the formal software cost estimating procedure.

the cost of development is divided into primary and secondary costs, thus:

$$C_D = C_P + C_S$$

where

C_D = Cost of Development

C_P = Primary Cost (Manpower)

C_S = Secondary Cost (Computer, Documentation, Etc.)

then,

$C_P = MM(C_e)$

where

MM = Total Development Man-Months

C_e = Average Labor Cost

and

$$C_S = \sum_{i=1}^{n} C_i = kC_p.$$

Therefore: $C_D = (MM)C_e(i+k)$

where

k = Ratio of Secondary to Primary Costs (=.075)

The total software development cost (does not include government Conceptual and Validation Phase costs) includes the costs of:

- Analysis
- Design
- Code
- Code
- Debug
- Test and Checkout

and is proportional to the total man-months of development effort.

Total Development Man-Months

This is the primary output variable. It is the basis for the total development cost estimate and it is the value from which the distribution of effort by life cycle phase is derived. The hours include those directly related to the development of the software system. They include the direct hours needed for:

Analysis - interpreting the system requirements and producing viable alternative system concepts

Design - preparing detailed designs of the data processing system and the individual programs

Coding and Debugging - writing individual modules and programs and performing individual tests

Testing and Checkout - integrating the individual subsystems into a complete system and conducting prescribed tests on the entire system.

The discussion of the model does not indicate the extent that support and management hours are included in the total. Also, there may be some question about the activities associated with concept development (e.g., is the test plan furnished by the government following the validation phase or is it developed as part of the project). As in many cost estimating situations, the line between concept analysis and the evaluation of solutions to selected concepts is hazy.

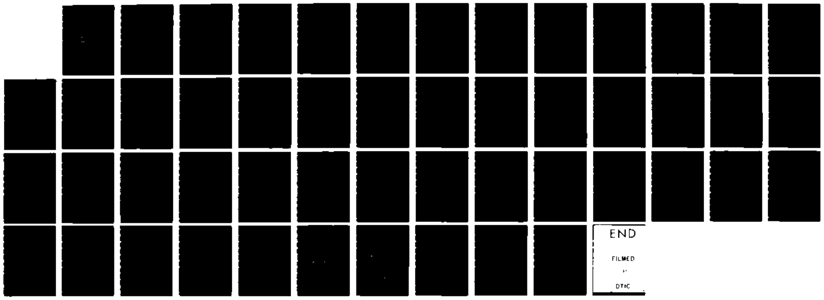
Although the DAI documentation and discussions with the authors indicate that the model includes integrated system testing, it appears that this effort is not included in the original SDC data which was the basis for the curve fits. (76% of the SDC data points describe programs that do not interface with any other programs).

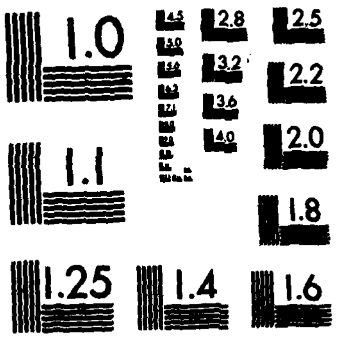
Software Development Time

A nominal development time is presented that implies "customary manloading". That is, the schedule does not reflect either crash projects or allow for unnessary delays.

AD-A123 818 PROJECTION OF MAXIMUM SOFTWARE MAINTENANCE MANNING LEVELS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA S L TODY ET AL. JUN 82 2/2

UNCLASSIFIED F/G 9/2 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

Distribution of Development Effort

The expenditure of time and effort associated with major project milestones is given for small projects (one level of supervision) and large projects (more that one level of supervision). The distributions are for nominal projects and do not allow for any possible acceleration or delay of the completion of the project....

Inputs

Program Size

DAI has been very carefull to describe the size variables which are the primary inputs to the estimates using the relationships. However, we should point out that the respondents to the original SDC questionnaire were not so well directed and it may be necessary when analyzing the structure of the model as it relates to prediction accuracy that significant errors may have been introduced by this failure to be specific. The DAI model may not overcome what are inherent limitations in the data.

The DAI procedure calls for several estimates in support of the DSARC process. It recognizes that the best estimates of program size are obtained later in the development cycle. It suggests, then, that the interpretation of the program size changes during the life cycle and that associated with the change are increases in estimating accuracy. The report describes how the knowledge of the size estimator changes during the life cycle and how this affects the estimating precision. The precision associated with the different size measures during the system development life cycle is as follows.

Code that is developed as part of the project but is not delivered to the customer is a source of variation in the estimate of the system size and must be considered. However, no guidance is provided for making any adjustment other than citing that the SDC data showed delivered code to average 77 percent of the developed code with a standard error of 30 percent.

SOFTWARE ESTIMATE	WHEN	SIZING BASIS	% ERROR
1. INITIAL PROGRAM BUDGETARY ESTIMATE	CONCEPTUAL PHASE	TOTAL OBJECT CODE	UP TO 200%*
2. INDEPENDENT PROGRAM VALIDATION COST ESTIMATE	VALIDATION PRIOR TO RFP RELEASE	TOTAL OBJECT MINUS DATA AREAS	UP TO 100%
3. INDEPENDENT FSD COST ESTIMATE	COMPLETION OF SYSTEM SPEC THROUGH PDR	TOTAL OBJECT MINUS DATA AREAS WITH ADJUSTMENTS FOR	UP TO 75%
4. UPDATE OF FSD COST ESTIMATE	PDR THROUGH REMAINDER OF DEVELOPMENT	TOTAL SOURCE CODE	UP TO 50% IMPROVING TO ZERO AT COMPLETION

*THE ACTUAL MAY BE 200 PERCENT OF THE ESTIMATED OR THE ESTIMATED MAY BE 200 PERCENT OF THE ACTUAL.

Allowance must also be made for support software development especially when working with new hardware.

Total Object Words

During the Conceptual Phase when very little is known about the system to be developed, the initial estimate is made using the analyst's judgement (usually by analogy with previously developed systems, but other methods are possible) of the number of object words occupied by "ever program needed to run and maintain the system in the field". This measure is obtainable from listings of computer system routines that build executable programs from the output of the compiler. Taking values from systems similar to the one being planned can provide a basis for estimating the value. Care should be taken, however, when program overlays are involved. Also, extensive use of standard library routines can greatly increase the words of object program size and not be representative of a comparable increase in development effort.

Total Object Words Minus Data Areas

The memory space occupied by an executable program is composed of locations containing instructions and locations reserved for the data upon which the program will operate. Sometimes the data storage areas are significantly larger than the area occupied by the actual instructions. DAI suggests that the effort required to develop the programs is more closely related to the size of the instruction space than to the size of the combined data and instruction storage. However, as in the case of the total object words, there is no evidence of this distinction being made in the original derivation of the estimating procedures. Also, there is no guidance provided on how to apply the additional information when preparing cost estimates. Some computer system executive processing routines provide this information. However, many don't and, therefore, it would be very difficult to obtain comparable historical information to guide new estimates.

New Object Words Minus Data Areas

Only the writing of new code contributes to the software development effort (if code written to modify existing modules is counted as new code). To account for the work done to adapt existing code to a new system, which includes analyzing the code and deciding how to modify it, any existing module that will result is less than 50 percent utilization of existing code is considered to be entirely new.

New Source Lines

Counts of new source lines written (whether in a higher order or machine oriented language) can be obtained from compiler listings, measuring card decks or text editors. It is one of the easiest measures of size to obtain. As in the previous case, modules containing less than 50 percent reused code are considered to be new.

Development Environment

For estimates made using lines of source code where the size is less than 10,000 lines, the estimating relationship includes a number of factors describing the development environment. These are included in the estimate when the indicated item is to be part of the development process....

f1 Special Display
f2 Detailed Definition of Operational Requirements
f3 Change to Operational Requirements
f4 Real Time Operation
f5 CPU Memory Constraint
f6 CPU Time Constraint
f7 First SW Developed on CPU
f8 Concurrent Developed on CPU
f9 Time Share Verus Batch Processing in Development
f10 Developer Using Computer at Another Target Computer
f11 Development at Operational Site
f12 Development Computer Different from Target Computer
f13 Development at More than One Site
f14 Programmer Access to Computer

After analyzing the method used by DAI to obtain their estimating relationships and after comparing their definitions of input and output variables with the original sources of data, it is clear that there are discrepancies between the way the data are being applied and what they originally represented. DAI does not explicity justify their approach but their presentation of the estimating procedure does give consideration to errors arising from differing definitions of the variables.

DAI seems to be saying that consistent use of the estimating procedures regardless of how they were obtained will produce results with at least a predictable error. That is, knowing the range of error that can occur because of differences in definitions and ability to predict the input variables will, when applied to the given estimating relationships, produce estimates with precision that is in accordance with previous experience. DAI further substantiates the approach of throwing all the error into the ability to define the input by presenting standard error values for the size variables at different times in the life cycle.

E. FARR AND ZAGORSKI MODEL

Description of the Model

System Development Corporation completed several projects for the Air Force, Electronic Systems Division in which they attempted to develop methods for predicting the cost of software development. The Farr and Zagorski model represent an intermediate stage in the program.

Using historical data from internal projects and from other organizations, the SDC team systematically tested over 100 variables to learn if they were satisfactory predictors of program design, coding and debugging effort.

Farr and Zagorski published three equations which were determined to be the best predictors tested up to that time.

$$MM = 2.7X_1 + 121X_2 + 26X_3 + 12X_4 + 22X_5 - 497 \quad (1)$$

$$MM = 2.8X_6 + 1.3X_7 + 33X_3 - 17X_8 + 10X_9 + X_{10} - 188 \quad (2)$$

$$MM = 8.4X_{11} + 1.8X_{12} + 9.7X_3 - 3.7X_{13} - 42 \quad (3)$$

Definition of Output

MM is the number of manmonths needed to design , code and debug a single program. The effort begins when a programmer or analyst is given a complete operational specification for a program and it ends when the program is released for integrated system testing.

Definitions of Inputs

X_1 = number of instructions in original estimate (in thousands)

X_2 = subjective rating of information system complexity (scale 1-5)

X_3 = number of document types delivered to customer

X_4 = number of document types for internal use

X_5 = number of computer words needed to store program data ($\log_{10}$)

X_6 = number of instructions in delivered program (in thousands)

X_7 = number of man-miles for travel (in thousands)

X_8 = system programmer experience (average of total years of experience with the computer, language, and application)

X_9 = number of display consoles

X_{10} = percent of instructions new to this program (not re-used from preveios versions)

X_{11} = number of instructions to perform decision functions (in thousands)

X_{12} = number of instructions to perform nondecision functions (in thousands)

X_{13} = programmer experience with this application (average number of years).

F. WOLVERTON

Description of the Model

Estimates of routine size are converted to costs using cost per instruction values that are functions of the routine type and complexity. The costs are fully burdened and when summed for all the system routines represent the total system development cost. Development extends from analysis and design through operational demonstration. A matrix of ratios is used to allocate the total cost to 7 phases with each phase divided into up to 25 activities. This allocation is compared from the standpoints of staff, schedule, and general credibility.

The model, then, is a combination of formal algorithm and judgement. It has been used successfully at TRW. As described by Wolverton, it features a data base of historical data that provide the necessary cost per instruction and allocation values. The procedure is adaptable to any new environment by creating a new data set representing local definitions of phases and activities and burdened cost conventions. In fact, Wolverton cautions that the given values cf cost per instruction are for illustration and users should prepare their own values.

TRW has computerized the maintenance of the cost data base and the allocation process. Given the inputs of size and complexity, the system calculates the cost allocations and facilitates any subsequent adjustments. Since most models are used in a similar manner, even if the procedure for using the model does not say so, there should be no compromise of the model's performance if the evaluation is based on a single estimate of costs. Other adjustments that are necessary to execute the model in different environments will be discussed later.

The estimating procedure begins by identifying all the routine comprising the system. Each routine size, category, and relative degree of difficulty are estimated by knowledgeable persons.

The categories that have "stood the test of usage" at TRW are:

- Control routine
- Input/Output routine
- Pre or Post algorithm processor
- Algorithm
- Data Management routine
- Time-Critical processor

Relative difficulty is indicated by six levels depending on whether a routine is Old or New and then by simply: Easy, Medium or Hard.

....Multiplying the cost per instructin for each routine by its number of object instructions and summing the products for all the routines yields the estimated total development cost.

The development cost is allocated to the following 7 phases using proportions for each phase that were obtained from the historical data base.

A. Performance and Design Requirements
B. Implementation Concept and Test Plan
C. Interface and Data Requirements Specification
D. Detailed Design Specification
E. Coding and Auditing
F. System Validation Testing
G. Certification and Acceptance Demonstration

Then, the cost for each phase is divided into up to 25 activities....

A matrix of computer hours by phase and software type is used to estimate computer usage costs for development.

<u>Outputs</u>

<u>Development Cost</u>

The given cost values are in 1972 dollars. The value of cost results from applying "bid rates" to labor costs which accounts for fringe benefits, overhead, administrative expenses and other indirect costs. Documentation and travel costs are added to the labor costs. Finally, estimates are made of the computer costs. The distribution of the costs by phases and activities were described above.

<u>Development Effort</u>

Cost is not a suitable basis for evaluating the different software estimating models because of differences in accounting practices among organizations and because of inflation. Therefore, the Wolverton cost values were converted to manmonths using an average burdened cost per manmonth of $4600. This value was obtained from the article describing the TRW estimating procedure and, therefore, should be representative of the cost environment.

<u>Inputs</u>

<u>Object Instructions</u>

The model input measure of size is applied to programs or routines. These are taken to be functionally distinct elements of a system that would be developed independently then intergrated into the delivered system. It is expected that these would be independently operable using test drivers. Such a definition is consistent with industry usage. The reference document is not specific on this point. The term "instructions" is taken literally. This means estimating the number of instructions in the executable program exclusive of any data areas. The number of instructions may be estimated by obtaining the words of memory occupied by the executable code and dividing by the average words per instruction.

<u>Software Categories</u>

Each routine is characterized according to one of the following categories:

C. Control Routine. Controls execution flow and is nontime critical.

I. Input/Output Routine. Transfers data into and out of computer

P. Pre-or Post Algorithm Processor. Manipulates data for subsequent processing or output.

A. Algorithm. Performs logical or mathematical operations.

D. Data Management Routine. Manages data transfer within the computer.

T. Time Critical Processor. Highly optimized machine dependent code.

Degree of Difficulty

Wolverton indicates that any numeric representation of complexity may be used. The main purpose is to distribute the cost per instruction values over the range of experience for a given category of software. He suggests a simple designation of old or new, depending on a loose interpretation of the amount of reusable code, and easy medium or hard compared with other programs in the same category.

APPENDIX B

ANALYSIS OF SOFTWARE MODELS BY WOLVERTON

A. INTRODUCTION

R. W. Wolverton studied several software cost estimating models while working for TRW in an effort to determine that model which would best predict those costs associat with software development. This appendix consists of e erpts from his review of some of these models.

B. BOEING COMPUTER SERVICE COST MODEL

Purpose

Boeing Computer Services (BCS) designed this analytical model to provide an estimate at proposal preparation time of the number of manmonths needed to design a computer program. BCS developed the model for use as an internal guideline to cross-check the traditional bottom-up estimate made by their proposal manager. The bottom-up estimate, with its WBS was tacitly assumed to be more accurate and the model served to aid in independently justifying the proposal manager's estimate.

While under contract to RADC, Boeing used their cost model to test several hypotheses about the cost benefit attributable to modern programming practices (Black, et al., 1977; Black, 1978). BCS derived and calibrated their model against internal software projects using traditional programming practices. This model has received wide-spread exposure as part of the DOD's embedded computer resources DSARC guidebook (DeRoze, 1977).

Input

a. Size of computer software in units of delivered source statements. The BCS model assumes that a "statement" is one fully checked tested, and documented statement coded in a selected language. The choice of high-level language can have a significant effect on the development cost, but ordinarily affects only portions of the total task.

b. Type of software to developed. BCS observed some combination of five generic functions. Each "type" has its own group productivity rate. The specific software type and productivity rates are as follows:

• Mathematical Opns	6 manmonths/ 1000 source statements
• Report Generation	8 manmonths/ 1000 source statements
• Logic Operations	12 manmonths/ 1000 source statements
• Signal Processing, Data Reduction	20 manmonths/ 1000 source statements
• Real-Time, Executive or Avionics Interfacing	40 manmonths/ 1000 source statements

The decreasing productivity is caused by the increasing complexity of the type of software being developed.

c. Tasks to be accomplished in the computer software development, are distributed by the BCS model as follows:

Task	% Total Cost
• Requirements Definition	5
• Design and Specification	25
• Code Preparation	10
• Code Checkout	25
• Integration and Test	25
• System Test	10

The numerical distribution opposite the task does not consider reuse and sophisticated debug tools. The distribution is not necessarily a rectilinear function of time, but is intended to be used as a guideline for schedule preparation. Documentation is not included in this estimating procedure and must be estimated by some other method, not defined in the model itself, and added to the manpower estimates.

d. Adjustment of the labor estimates is accomplished by means of table lookup multipliers given in Table VIII. All terms are assumed by the model developer to be self-explanatory.

<u>Computational Procedure</u>

Using this model, Program Office personnel would estimate how much of the total OFP software is closest represented by one of the five generic types of software. In practice, estimating the size and type would be based on past experience with similar projects that have been adjusted to the new application. Everything associated with the manmonth estimate flows from this first step.

Table VIII provides the estimator with phase-sensitive multipliers for adjusting the baseline manmonths estimate. The user should be alert to stringent sizing or timing limitations. These effects should be estimated by some other procedure (not given) and added to the baseline manmonth estimate.

After individual labor costs have been adjusted by use of the table, the BCS model sums up the individual estimates and arrives at the total labor cost for the project. Computer time is estimated by a rule of thumb that approximately three hours of stand-alone computer time will be spent per manmonth.

TABLE VIII

LABOR ESTIMATE ADJUSTMENT FACTOR

	REQUIREMENTS DEFINITION	DESIGN AND SPECIFICATION	CODE PREPARATION	CODE CHECKOUT	INTEGRATION AND TEST	SYSTEM TEST
REIMPLEMENTATION OF EXISTING SOFTWARE	0.2	0.2	0.8		0.8	
FOLLOW-ON CONTRACT WITH CURRENT CUSTOMER	0.7					0.9
NUMBER OF PROGRAMMERS: 1-2	0.2	0.5	0.8		0.2	
(INTERPOLATE BETWEEN 6-10	1.0	1.0	1.0		1.0	1.[illegible]
VALUES IF NEEDED) MORE THAN 20	6.0	3.3	1.2		3.0	3.[illegible]
HIGHER ORDER LANGUAGE (SEASONED COMPILER)		0.3	0.3	0.2		
MACRO-LANGUAGE - IN CODING		0.9	0.9	0.9		
- FORMS FOR DOCUMENT		0.8			0.8	
ON-LINE CODE/DATA ENTRY			0.9	0.9		
ON-LINE DEBUGGING				0.6		
POOR (OR NO) DEBUG TOOLS EXCEPT DUMPS				1.4	1.4	
PROGRAMMING EXPERIENCE WITH ENGINEERING/TECHNICAL DISCIPLINE OF APPLICATION: ENTRY LEVEL	2.0	3.0	1.5			1.5
- MODERATE	1.0	1.0	1.0			1.0
- HIGH	0.6	0.5	0.8			0.7

Output

The fundamental output is the total manmonths estimated for the planned software project. In turn, the total manmonths are spread over a six stage development cycle from requirements definition to system test.

Although acceptable engineering accuracy in estimating total manmonths is claimed by the model developers for traditional programming practices (c. 1970), the examples of estimating accuracy are not encouraging for modern programming practices. In other words, the intent of the BCS model is to show how much a new project would have cost if done the old way. Presumably the lower observed cost is due to the new design methodologies. Output results for five projects given by BCS are shown in Table IX. A guideline is to try this model on some historical data and compare the accuracy of predicted versus actual manmonths before attempting to use it in practice....

TABLE IX

Forecasted versus Actual Costs for the BCS Model

Project	Forcast Total Manmonths	Actual Total Manmonths	Forecast/Actual Ratio
A	419.7	71.0	5.9
B	2288.5	991.7**	2.3
C	51.5	43.8	1.2
D	3298.7	514.8**	6.4
E	7.9	7.3	1.1

** Contains some estimate-to-complete data, along with actuals

C. IBM WALSTON-FELIX COST MODEL

Purpose

Walston and Felix conducted experiments on 60 completed software development projects in their search for a method of estimating programing productivity (Walston-Felix, 1977). The purpose of this effort was to estimate the rate of production of lines of code by projects, as influenced by project conditions and requirements.

Five specific objectives of the Walston-Felix model are

a. To evaluate improved programming technologies.

b. To provide support for proposals and contract performance.

c. To gather historical records of the software development work performed.

d. To provide programming data to management.

e. To foster a common programming terminology.

Completed projects in the Walston-Felix data base ranged in size from 4,000 to 467,000 delivered source lines of code and in effort from 12 to 11,758 manmonths. Applications programs included realtime process control; interactive, report generators; data base control; and message switching programs. Twenty-eight different high-level languages and 66 different computers are represented in their data base. This is an outstanding example of a closed-form model obtained by linear regression analysis of a large and diverse body of actual software projects. Some further technical work is required to extend the findings of Walston and Felix to the specialized needs of avionics software. The additional work to be done in calibration of the model will be discussed in...Computational Procedure.

Input

a. Number of lines of delivered source code. Source lines are 80-character source records provided as input to a language processor. Job control languages, data definitions, link edit language, and comment lines are included. Reused code is not included.

b. From the raw data provided by the 60 projects, a set of 68 variables was selected for analysis to find which ones were significantly related to productivity. Twenty-nine of the variables showed a significant correlation with productivity and have been retained for use in estimating....

c. ....The model user is asked to answer a multiple-choice question in his response to the statement: User participation in definition of requirements is: none, some, much. In the origional analysis the mean productivity was computed for the 60 completed projects for which no user participation was reported and found to be 491 DSL/MM. The mean productivity for all projects that reported some user participation was 267 DSL/MM, and the mean productivity for those reporting much user participation was 205 DSL/MM. The absolute value of the change in productivity from no user participation to much user participation is found to be 286 DSL/MM....

Computational Procedure

The Walston-Felix cost model can aid Program Office personnel in estimating five project parameters: productivity, schedule, cost, quality, and size of the software product to be delivered. One difficulty is in identifying and measuring independent variables that can be used to estimate the desired variables, such as estimating the size of the software product to be delivered. We take the point of view that the size of the software product to be delivered can be independently, albeit with difficulty, estimated from the internal historical data base associating avionics function with size (Battelle, 1978) or avionics function with software requirements (Heninger, et al., 1978).

Productivity is a significant variable in all software estimating processes. Programming productivity is defined here as the ratio of the delivered source lines of code (DSL) to the total project effort in manmonths (MM) required to produce the delivered product. Total manmonths covers the management, administration, analysis, operational support, documentation, design, coding, and testing effort expended in the development phase. Analytical results are derived at start of work, PDR, midway through software development, at acceptance test completion, and every three months during the service or maintenance phase.

The 29 variables...are combined into an index based on the effect of each variable on productivity from previous analysis. The productivity index is computed as follows:

$$I = \sum_{i=1}^{29} W_i X_i$$

where

I	= productivity index for a project
W_i	= question weight, calculated as $0.5 \log_{10}(PC)_i$
$(PC)_i$	= productivity change indicated for a given question i....
X_i	= question response (+1, 0, or -1), depending on whether the response indicates increased, nominal, or decreased productivity.

....The data set is analyzed by ordinary least squares and the standard error of estimate, or standard deviation of residuals, is shown as dashed lines. In the data sample studied, the productivity index ranged from -4 to +4 (private communication with C. Walston). The Air Force model user would determine his own productivity index for a single project by answering the 29 questions...and by calculating I according to the above formula. He then multiplies his average productivity for all past avionics software in his data base by the productivity index for the acquisition at hand.

If the Program Office has a historical data base of many projects, the total effort can be determined by a least squares fit and the regression equation from the Program Office's own internal data analysis at the point $I = 0$, DSL/MM = 274, using the coordinate system.... A statistical analysis program such as the Statistical Package for the Social Sciences (a product of SPSS, Inc.) would be helpful. SPSS will also provide other descriptive statistics such as the standard error of the linear regression line....

The statistics...are given by medians and quartiles because of the variability in the measurement data. Note that the median productivity ($I = 0$) is 274 DSL/MM. The median for the size of the delivered software product is 20,000 lines; 50 percent of the projects reported that the size of their delivered code ranged from 10,000 to 59,000 lines. Resources for project development are shown. The error detection results are for the distribution of errors reported during the development period....

The amount of calendar time to allow for the development of software is difficult to express from a closed-form model. However, the equation for project duration in months as a function of total effort in manmonths was found to be:

$$M = 2.47\ E^{0.35}$$

where,

M = duration in months, for full-scale development

E = effort in manmonths, for full-scale development.

From the data collected for service projects, certain descriptive statistics were calculated.... The interpretation is the same as before: median data and quartile data are presented due to the scatter in the raw reports. No predictive relationships are given for service projects.

Documentation, as defined in this model, consists of program functional specifications and descriptions, users' guides, test specifications and results, flowcharts, and program source listings that are delivered as part of the documentation. To a close approximation, the least squares equation for the number of pages of delivered documentation varies directly as the number of lines of source code; that is

$$D = 49\ L^{1.01}$$

where,

D = pages of documentation, including source listings

L = thousands of source code lines

<u>Output</u>

The major outputs available to the model user are as follows:

a. Total effort in manmonths required to produce the lines of source code.
b. Duration of project in months.
c. Use of improved programming technologies expressed as a percentage of code developed using each technique.
d. Estimated productivity of project as influenced by project environment and requirements.
e. Pages of documentation for the intended project, including pages of source listings delivered as part of the documentation requirements.
f. The results do not support answers to certain project attributes implied by the data coefficients...because of cross-correlation effects (i.e.,

the individual attributes are not statisticlly independent). For example:

1. Chief programmer team.
2. Top down development.
3. Structured programming.
4. Design and code inspections.

The contribution of each attribute could not be taken individually because in the definition of chief programmer team the other techniques are implied.

g. Other descriptive statistics can be inferred from study of the report itself; for example, the cost of computing time and the average number of people (total manmonths of effort divided by the duration) as a function of the total effort. The responsibility of relating the lines of executable assembly code to lines of delivered source code rests with the model user.... A scaling law for the Walston-Felix model can be derived from internal avionics historical data.

D. PUTNAM'S SOFTWARE LIFE CYCLE COST MODEL (SLIM)

Purpose

A descriptive cost model, coupled with informed opinion, will aid in answering top-level management questions about the development of OFP software. Descriptive statistics associated with expected OFP software cost, development time, manning levels, and perturbations about these estimates are significant management interests at pre-RFP time. The Air Force can specify a useful lifetime, say 10 years, and obtain a quantitative cost estimate of the OFP software life cycle subject to the assumptions of the model.

Input

Three input parameters are required to calibrate this model's technology constant (Ck) for avionics applications. The F-111 data point...was the basis for this calibration. The three data points are:

a. Number of delivered lines of executable source code, not including comments: 22,100.

b. Number of manmonths for developing software: 805.

c. Number of calendar months for developing software: 33.

The user is prompted for all inputs by the EDITOR built into the SLIM cost model. Seventeen on-line inputs required for this model are as follows:

a. Enter title of software system. Avionics, F-111

b. Enter start date (MMYY). 0174

c. Enter the fully burdened labor rate ($/MY) at your orgainization. 60000

d. Enter the standard deviation of your labor rate ($/MY). 6000

e. Enter the anticipated inflation rate as a decimal fraction. 0.065

f. Enter the proportion of development that will occur in on-line, interactive mode. 0

g. Enter the proportion of the development computer that is dedicated to this system development effort. 0.2

h. Enter the proportion of the system that will be coded in a HOL. 0

i. Enter the number corresponding to the primary language to be used. (Twelve choices are given.) 10 = assembly level language.

j. Enter the number corresponding to the type of your system. 1

   1. Real-time or time critical system

   2. Operating system

3. Command and control
4. Business application
5. Telecommunication and message switching
6. Scientific system
7. Process control.

k. Choose the response below which best describes your system. 2

1. The system is entirely new, with many interfaces, and must interact within a total management information system structure.
2. This is a new stand-alone system. It is simpler because the interface problem with other systems is eliminated.
3. This is a rebuilt system with large segments of existing logic. The primary tasks are recording, integration, interfacing, and minor enhancements.
4. This is a composite system made up of a set of independent subsystems with few interactions and interfaces among them. Development of the independent subsystems will occur as a considerable overlap.
5. This is a composite system made up of a set of independent subsystems with a minimum of interactions and interfaces among them. Development will occur in parallel.

l. Enter the the proportion of memory of the target machine that will be utilized by the software system. 0.85

m. Enter the proportion of real-time code. 1

n. Below is a set of modern programming techniques that may be used on a software development project. Beside each are three possible responses indicating the degree of usage on your system. 1

Technique	Response
Structured Programming	1) < 25%
	2) 25-75%
	3) >75%
Design and Code Inspection	1) < 25%
	2) 25-75%
	3) > 75%
Top-down Development	1) < 25%
	2) 25-75%
	3) > 75%
Chief Programmer Teams	1) <25%
	2) 25-75%
	3) >75%

o. Below are two indicators of personnel that can impact the cost and time to do a project. Beside each are three possible answere indicating the degree of experience. 2

Personnel Experience	Response
Overall Skill and Qualification	1) Minimal
	2) Average
	3) Extensive
With Development Computer	1) Minimal

p. Enter sizing information in one of two forms:
1. An overall range of sizes, or
2. Ranges of size on a module-by-module basis.
Enter 1 or 2 to indicate how sizing data should be entered. 1

q. Enter the lowest possible and highest possible size in source statements. 18100, 26100

Computational Procedure

Total effort can be determined from the software equation developed by L. H. Putnam (Putnam, 1978; Putnam and Fitzsimmons, 1979). The software equation is modified by the environmental input parameters, items f through o. The software equation is:

$$S_s = C_k K^{1/3} t_d^{4/3}$$

where,

S_s = number of delivered lines of executable source code, not including comments

C_k = a state of technology constant; previous experience with computer response times and progamming practices gives:

C_k = 754 for avionics, assembly-level language

C_k = 4984 for "1973-style" arbitrary development

C_k = 10040 for "1979-style" structured development.

K = Rayleigh/Norden life cycle effort parameter in units of manmonths or manyears

t_d = Rayleigh/Norden time parameter. Time at which peak manpower nominally occurs for large software projects. Mathematically, it is the peak of the curve,

$$Y' = K/t_d^2 \; t e^{-t^2/2t_d^2}$$

K/t_d^2 = system difficulty, or ratio of total effort to development time squared.

The software equation is used to obtain engineering quality estimates during the early phases of a software project. The software equation is solved using a gradient constraint, $K = VD\, t_d^3$, where the magnitude of the difficulty gradient is empirically found for a particular development environment. Monte Carlo simulation is used to generate descriptive statistics associated with the effort, development time, and development cost. The standard deviations are used in calculating risk profiles.

The effort, time, and cost point estimates can be presented in the form of probability plots assuming a gaussian distribution. All that is needed is an extimate of the expected value (plotted at the 50 percent probability level) and the standard deviation (plotted offset from the expected value at the 16 percent probability level) to generate the probability line on ordinary probability paper. Then one can determine for example, that there is a 90 percent probability that the software development will not take more than x-manmonths of effort. When repeated for all probability levels of interest, one has a risk profile for that estimate.

The tradeoff law can be obtained from the software equation by solving for K. With a Monte Carlo simulation for generating variances for K and td one can perform a tradeoff analysis, pick a reasonable effort (or cost) time combination and complete the sensitivity analysis. The value of simulating several thousand Monte Carlo runs is that it produces a measure of the variation in effort and development time, or the risk profile. Knowing the sensitivities, the Air Force PM can use it effectively in planning and contracting so that the risk level is always within acceptable range. Examples of this procedure are given in the COMPSAC 77 tutorial (Putnam and Wolverton, 1977).

Output

Three options are available to the user: calibrate, editor, estimate. The option chosen for this illustration was "estimate." A file is built from the previous input data, and an on-line comment shows that the input data check was acceptable. The structure of the on-line output is shown below:

a. Summary of input parameters: table of all inputs. Annotated comment shows Ck, the technology constant, was separately computed to be 754.

b. Simulation: system cost summary is given as follows:

	Mean	Std Dev
System Size (STMTS)	22100.0	1333.0
Minimum Development time (Months)	34.8	1.2
Development Effort (Manmonths)	891.0	106.9
Development Cost (x $1000)		
- Uninflated dollars	4461.0	711.0
- Inflated dollars	4887.0	787.0

c. Sensitivity profile for minimum time solution (i.e., expected values of time, effort, and cost for the whole size profile):

	Source Statements	Months	Man-Months	Cost (x $1000)
-3 SD	18100	31.9	525	2627
-1 SD	20767	33.9	763	3814
Most Likely	22100	34.8	891	4461
+1 SD	23433	35.6	1034	5172
+3 SD	26100	37.3	1331	6657

Where SD = Standard Deviation

d. A cross-check with data from other systems of the same size for the most likely estimates is given. As compared with the RADC data base (which is a mixture of software projects), the remarks show less than normal productivity for avionics OFP software. This is to be expected.

e. An on-line information note gives the user 14 options for the remaining output; several of these will be given to show the management parameters available.

f. Linear program: this function uses the technique of linear programming to determine the minimum effort (and cost) or the minimum time in which a system can be built. The results are based on the actual manpower, cost, and schedule constraints of the user, combined with the system constraints provided earlier.

1. Enter the maximum development cost in dollars. 4500000
2. Enter maximum development time in months. 36
3. Enter the minimum and maximum number of people allowed on board at peak manloading time. 15, 40

	Time	Effort	Cost (x $1000)
Minimum Cost	36.0 Months	778 MM	3892
Minimum Time	34.8 Months	889 MM	4446

g. A tradeoff analysis within these limits is shown below.

Time	Manmonths	Cost (x $1000)
34.8	889	4446
35.0	869	4345
35.2	849	4247
35.4	830	4152
35.6	812	4059
35.8	794	3970
36.0	778	3892

h. Front end estimate: recall that the SLIM model assumes that the estimated time length is from logic design. Therefore, a separate front end estimate is required, as follows:

	Time (months)			Effort (MM)		
	(L)	(E)	(H)	(L)	(E)	(H)
Feasibility Study	7.8	8.7	9.6	9	35	61
Functional Design	10.4	11.6	12.8	25	50	75

Note: L = Low, E = Expected, H = High

i. Manloading: The table shows the mean projected effort and associated standard deviations required for development. The input parameters are

	Mean	Std Dev
Development Effort (Manmonths)	891.0	106.9
Development Time (Months)	34.8	1.2

Time	People/ Month	Std Dev	Cumulative Manmonths	Cumulative Std Dev
Jan 74	2	0	2	0
Feb 74	5	1	7	1
Mar 74	9	1	16	2
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
Oct 76	17	2	877	105
Nov 76	15	2	893	107
Dec 76	7	1	900	108

(This distribution of 36 rows is essentially a Rayleigh distribution over the calendar period of performance, with integer values for all entries.)....

o. Other primary outputs from the Slim cost model include:

1. Code production: calendar time versus cumulative source statements
2. Computer usage: calendar time versus CPU hours
3. Documentation: expected number of pages of documentation
4. Design-to-cost: SLIM has provided its best estimate of the minimum time and corresponding maximum effort (and cost) to develop your system. A greater effort would result in a very risky time schedule. However, if a lower effort is specified (within reasonable limits), development is still feasible as long as more time is allowed.
   Entered desired effort in manmonths. 805

	Mean	Std Dev
New Development Time (Months)	35.7	1.2
New Development Cost (x $1000)	$4025	488.0

5. The original file is updated with these new parameters, and the user can run manloading and cash flow or life cycle to see how these savings can be realized. This can be used interatively to match some projected benefit stream and get the project approved. (Connect time was about 37 minutes to run SLIM, at a cost of about $25)

In summary, the SLIM model is a descriptive, macro-level cost estimating tool applicable to OFP software, provided that its technology constant (Ck) is calibrated from valid historical OFP project data: number of delivered lines of executable source code; number of manmonths from project start to software acceptance; and number of calendar months for the development. This step and its consequences must be understood by the user. SLIM composes the feasibility study and functional design as a separate front-end estimate which must be added to the initial cost estimate. Labor mix and work breakdown structure information is not given. Resources are allocated against time (spread by a Rayleigh distribution), but not against function (e.g., analysis and design, code and debug, and test and integration). All statistical parameters are assumed to be normally distributed for mathmematical tractability. This assumption may contribute to the extreme sensitivity between minimum cost and minimum time as shown in item f, linear program example; i.e., a 3 percent change in calendar time (from 36 to 34.8 months) corresponds to a 14 percent change in cost ($3892K to $4446K). All mathematical expressions used in the computational procedure are continuous functions; therefore the model will always produce a calculated estimate. As with all models, this estimate must be tested against experience and human insight.

APPENDIX C

SUPPORTING DATA AND CURVES

TABLE X

Project A Data

Actual Manmths	Time Mths	Predicted LC Manmths	Predicted Maintenance Manmths
4.9600	1	2.6005	
5.4600	2	5.0970	
7.1380	3	7.3683	
9.1380	4	9.3453	
11.9180	5	10.9544	
12.1380	6	12.1522	
13.1380	7	12.9205	
12.1380	8	13.2663	
11.9240	9	13.2183	
15.2690	10	12.8235	
13.2800	11	12.1413	
9.8460	12	11.2388	
8.3077	13	10.1846	
10.8460	14	9.0446	
6.8460	15	7.8778	
5.8460	16	6.7342	
5.8460	17	5.6528	
3.0000	18	4.6616	1.19124
3.2800	19	3.7780	2.22748
2.8400	20	3.0101	2.98686
4.0000	21	2.3583	3.40398
3.0000	22	1.8174	3.47740
2.0000	23	1.3778	3.26075
2.0000	24	1.0278	2.84229
2.0000	25	0.7545	2.32054
2.0000	26	0.5451	1.78318
2.0000	27	0.3877	1.29398
2.0000	28	0.2715	0.88884
1.5000	29	0.1871	0.57894

TABLE XI

Project B Data

Actual Manmths	Time Mths	Predicted LC Manmths	Predicted Maintenance Manmths
5.9200	1	3.8688	
5.9200	2	7.4128	
7.8600	3	10.3523	
13.4200	4	12.4888	
15.8000	5	13.7264	
15.5800	6	14.0751	
14.3400	7	13.6363	
13.1800	8	12.5768	
12.0200	9	11.0966	
5.0000	10	9.3971	
4.3333	11	7.6564	1.76084
2.7500	12	6.0121	3.32648
4.5556	13	4.5561	4.53730
4.4722	14	3.3355	5.29599
5.4167	15	2.3610	5.57900
5.5000	16	1.6169	5.43158
5.6111	17	1.0719	4.94937
3.7778	18	0.6882	4.25312
3.8889	19	0.4280	3.46350
2.7778	20	0.2580	2.68174
1.5833	21	0.1508	1.97898

TABLE XII

Project C Data

Actual Manmths	Time Mths	Predicted LC Manmths	Predicted Maintenance Manmths
6.0	1	2.8213	
7.5	2	5.5154	
7.0	3	7.9644	
8.5	4	10.0687	
12.5	5	11.7533	
12.5	6	12.9721	
13.0	7	13.7095	
14.0	8	13.9789	
14.0	9	13.8191	
14.0	10	13.2888	
13.0	11	12.4601	
11.0	12	11.4116	
11.0	13	10.2221	
8.0	14	8.9650	
8.0	15	7.7044	
9.0	16	6.4920	
3.0	17	5.3669	0.64025
2.0	18	4.3546	1.25743
2.0	19	3.4692	1.82983
2.0	20	2.7146	2.33840
2.0	21	2.0868	2.76779
2.5	22	1.5764	3.10709
4.0	23	1.1704	3.35022
3.0	24	0.8543	3.49602
4.0	25	0.6131	3.54788

TABLE XIII

Project D Data

Actual Manmths	Time Mths	Predicted LC Manmths	Predicted Maintenance Manmths
6.0000	1	3.8585	
9.5200	2	7.1746	
8.5769	3	9.5312	
9.6369	4	10.7213	
9.6369	5	10.7702	
11.1700	6	9.8940	
10.2260	7	8.4176	
5.2800	8	6.6828	
1.6800	9	4.9749	0.66747
2.4800	10	3.4844	1.31143
3.0000	11	2.3015	1.90977
3.0000	12	1.4316	2.44297
3.0000	13	0.8477	2.89525
5.0000	14	0.4738	3.25522
3.5000	15	0.2510	3.51641
2.5000	16	0.1261	3.67722
3.0000	17	0.0601	3.74074
4.0000	18	0.0272	3.71413
2.0000	19	0.0117	3.60786
3.0000	20	0.0048	3.43475
3.5000	21	0.0019	3.20901
2.0000	22	0.0007	2.94528
2.7600	23	0.0002	2.65777
3.0000	24	0.0001	2.35956
2.5000	25	0.0000	2.06207
1.5000	26	0.0000	1.77470
1.0000	27	0.0000	1.50475
1.5000	28	0.0000	1.25734
1.5000	29	0.0000	1.03565
1.0000	30	0.0000	0.84109
1.0000	31	0.0000	0.67365
1.0000	32	0.0000	0.53218
1.0000	33	0.0000	0.41475
2.0000	34	0.0000	0.31892

TABLE XIV

Combined Project A-D Data Normalized to td=1

Actual Manmths	Time Mths	Predicted LC Manmths	Predicted Maintenance Manmths
4.9600	0.100	2.3128	
6.0000	0.111	2.5637	
6.0000	0.167	3.8213	
5.4600	0.200	4.5433	
5.9200	0.200	4.5433	
7.5000	0.222	5.0151	
7.1380	0.300	6.6140	
7.0000	0.333	7.2503	
9.5200	0.334	7.2692	
9.1380	0.400	8.4568	
5.9200	0.400	8.4568	
8.5000	0.444	9.1807	
11.9180	0.500	10.0167	
8.5769	0.501	10.0307	
12.5000	0.555	10.7390	
12.1380	0.600	11.2541	
7.8600	0.600	11.2541	
12.5000	0.666	11.8826	
9.6369	0.668	11.8993	
13.1380	0.700	12.1468	
13.0000	0.777	12.5957	
13.4200	0.800	12.6900	
12.1380	0.800	12.6900	
9.6369	0.835	12.7992	
14.0000	0.888	12.8875	
11.9240	0.900	12.8950	
11.1700	1.000	12.7876	
15.2690	1.000	12.7876	
14.0000	1.000	12.7876	
15.8000	1.000	12.7876	
13.2800	1.100	12.4049	
14.0000	1.111	12.3478	
10.2260	1.167	12.0167	
9.8460	1.200	11.7921	
15.5800	1.200	11.7921	
13.0000	1.222	11.6313	
8.3077	1.300	10.9993	
11.0000	1.333	10.7069	
5.2800	1.334	10.6979	
10.8460	1.400	10.0777	
14.3400	1.400	10.0777	
11.0000	1.444	9.6444	
6.8460	1.500	9.0770	
1.6800	1.501	9.0667	
8.0000	1.535	8.7165	
5.8460	1.600	8.0424	
13.1800	1.600	8.0424	
8.0000	1.666	7.3605	
2.4800	1.668	7.3399	
5.8460	1.700	7.0134	
9.0000	1.777	6.2456	
12.0200	1.800	6.0224	

Table XIV continued

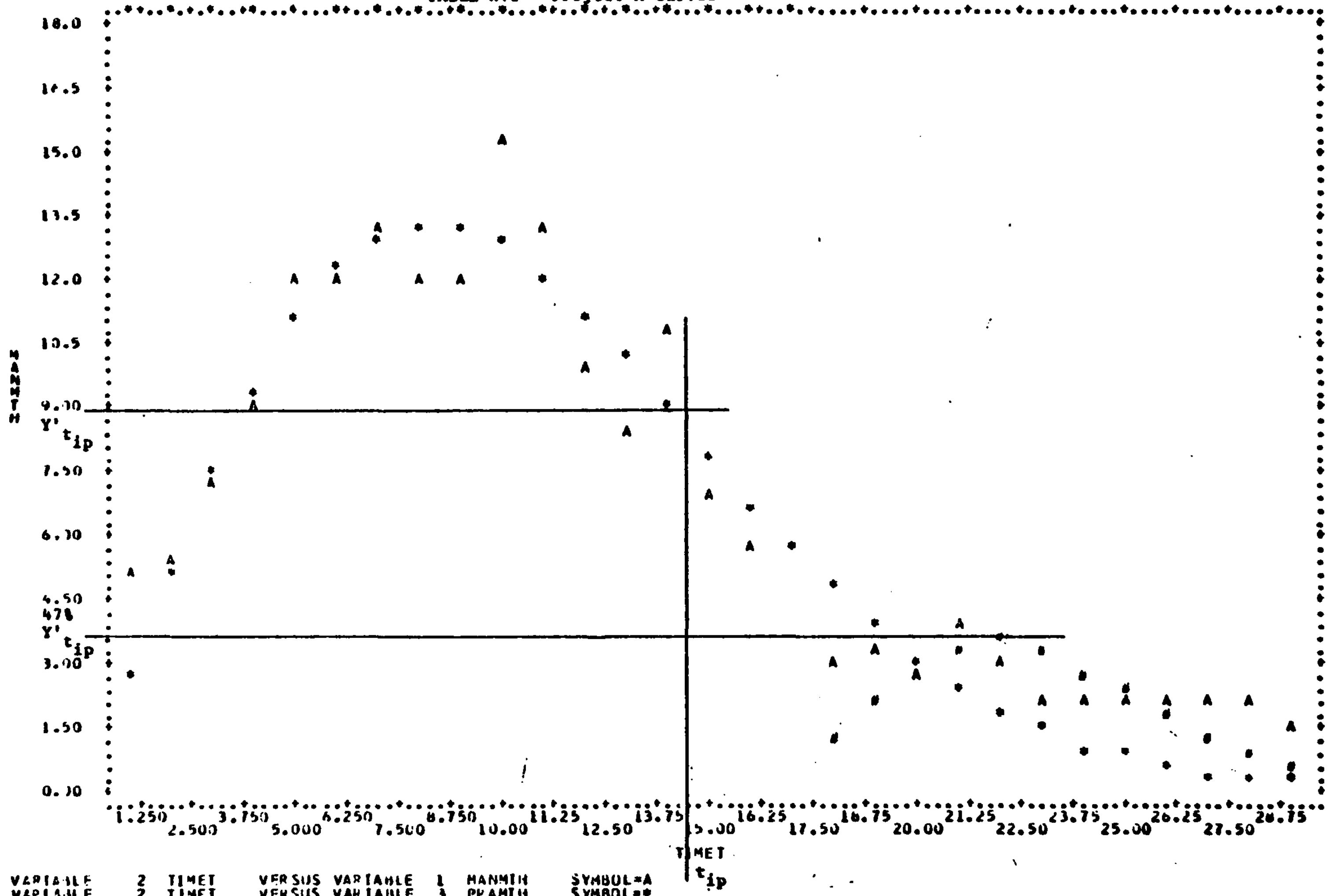
Actual Manmths	Time Mths	Predicted LC Manmths	Predicted Maintenance Manmths
3.0000	1.800	6.0224	0.00528
3.0000	1.835	5.6893	0.18457
3.0000	1.888	5.2016	0.46312
3.2800	1.900	5.0941	0.52590
3.0000	2.000	4.2458	1.04203
2.8400	2.000	4.2458	1.04203
2.0000	2.000	4.2458	1.04203
5.0000	2.000	4.2458	1.04203
4.0000	2.100	3.4880	1.53892
2.0000	2.111	3.4104	1.59202
3.0000	2.167	3.0332	1.85663
3.0000	2.200	2.8249	2.00771
4.3333	2.200	2.8249	2.00771
2.0000	2.222	2.6917	2.10625
2.0000	2.300	2.2559	2.44037
2.0000	2.333	2.0882	2.57399
5.0000	2.334	2.0832	2.57797
2.0000	2.400	1.7768	2.82995
2.7500	2.400	1.7768	2.82995
2.5000	2.444	1.5926	2.98621
2.0000	2.500	1.3803	3.17078
3.5000	2.501	1.3768	3.17393
4.0000	2.535	1.2595	3.27772
2.0000	2.600	1.0579	3.45858
4.5556	2.600	1.0579	3.45858
3.0000	2.666	0.8810	3.61804
2.5000	2.668	0.8761	3.62249
2.0000	2.700	0.7999	3.69053
4.0000	2.777	0.6392	3.83018
4.4722	2.800	0.5969	3.86530
2.0000	2.800	0.5969	3.86530
3.0000	2.835	0.5370	3.91294
1.5000	2.900	0.4395	3.98301
4.0000	3.000	0.3194	4.04514
5.4167	3.000	0.3194	4.04514
2.0000	3.167	0.1820	4.03290
5.5000	3.200	0.1622	4.01462
3.0000	3.334	0.1001	3.89258
5.6111	3.400	0.0782	3.80710
3.5000	3.501	0.0531	3.64877
3.7778	3.600	0.0358	3.46657
2.0000	3.668	0.0272	3.32901
3.8889	3.800	0.0156	3.04092
2.7600	3.835	0.0134	2.96117
2.7778	4.000	0.0065	2.57596
3.0000	4.000	0.0065	2.57596
2.5000	4.167	0.0030	2.18624
1.5833	4.200	0.0025	2.11088
1.5000	4.334	0.0013	1.81450
1.0000	4.501	0.0006	1.47364
1.5000	4.668	0.0002	1.17173
1.5000	4.835	0.0001	0.91255
1.0000	5.000	0.0000	0.69870
1.0000	5.167	0.0000	0.52270
1.0000	5.334	0.0000	0.38337
1.0000	5.501	0.0000	0.27572
2.0000	5.668	0.0000	0.19449

TABLE XV

NASA Project Data

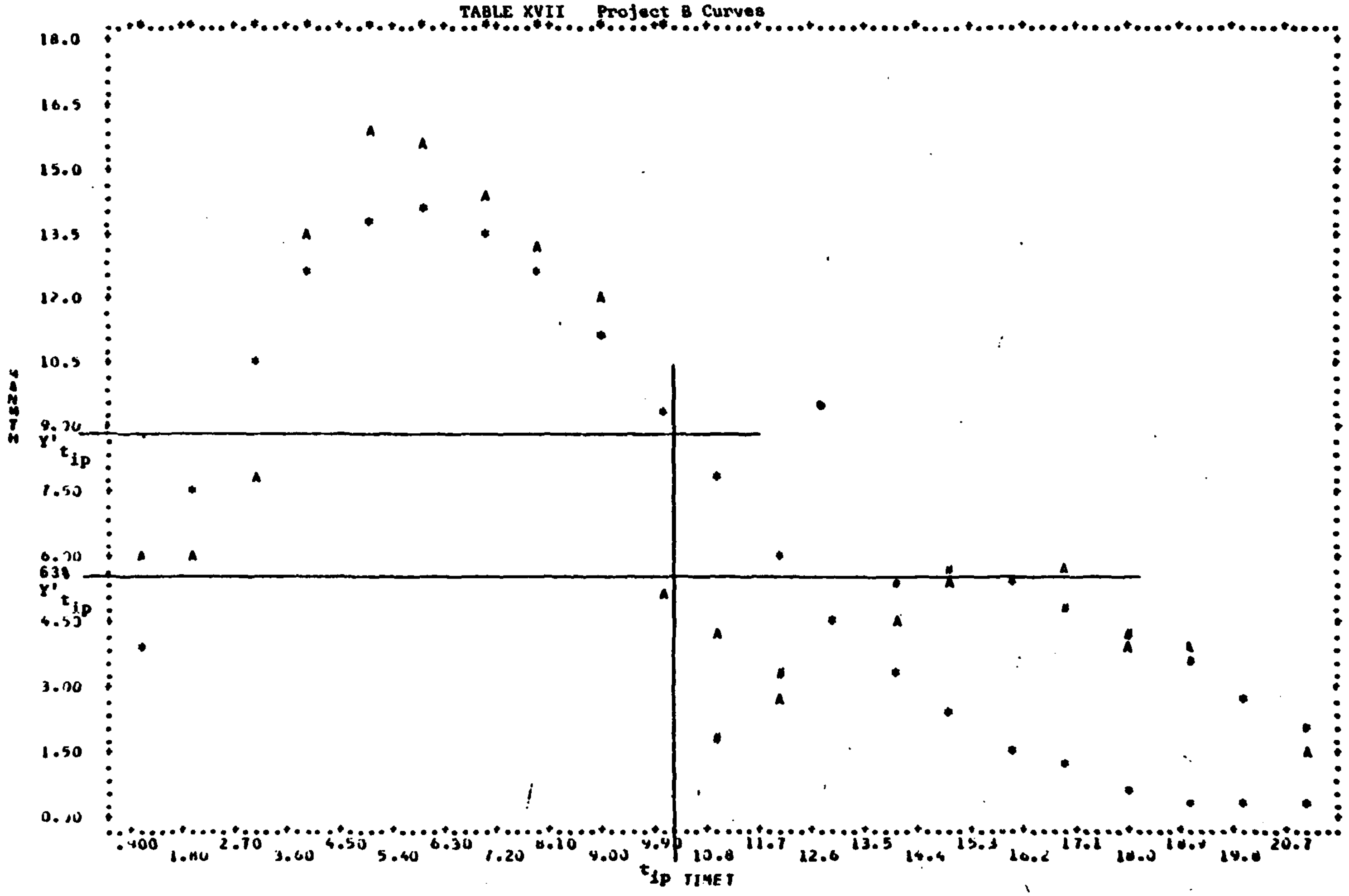
DATE	MHRS	MMTHS	MYRS	DATE	MHRS	MMTHS	MYRS
3/75	593	0.797	0.067	9/78	450	0.625	0.051
4/75	653	0.907	0.074	10/78	450	0.605	0.051
5/75	773	1.039	0.088	11/78	400	0.556	0.046
6/75	780	1.083	0.089	12/78	410	0.551	0.047
7/75	864	1.161	0.098	1/79	510	0.685	0.058
8/75	929	1.249	0.106	2/79	420	0.625	0.048
9/75	953	1.324	0.109	3/79	370	0.497	0.042
10/75	1013	1.362	0.115	4/79	410	0.569	0.047
11/75	1006	1.397	0.115	5/79	390	0.524	0.044
12/75	1037	1.394	0.118	6/79	440	0.611	0.050
1/76	1061	1.426	0.121	7/79	670	0.901	0.076
2/76	877	1.260	0.100	8/79	520	0.699	0.059
3/76	1150.5	1.546	0.131	9/79	580	0.806	0.066
4/76	1073	1.490	0.122	10/79	440	0.599	0.050
5/76	1055.5	1.419	0.120	11/79	294	0.408	0.034
6/76	1108	1.539	0.126	12/79	275	0.370	0.031
7/76	1000	1.344	0.114	1/80	410	0.551	0.047
8/76	867	1.177	0.100	2/80	367	0.527	0.042
9/76	640	0.889	0.073	3/80	541	0.727	0.062
10/76	422	0.567	0.048	4/80	482	0.669	0.055
11/76	340	0.472	0.039	5/80	299	0.402	0.034
12/76	260	0.349	0.030	6/80	449	0.624	0.051
1/77	188	0.253	0.021	7/80	418	0.562	0.048
2/77	290	0.432	0.033	8/80	216	0.290	0.025
3/77	444	0.597	0.051	9/80	214	0.297	0.024
4/77	390	0.542	0.044	10/80	230	0.309	0.026
5/77	280	0.376	0.032	11/80	361	0.501	0.041
6/77	320	0.444	0.036	12/80	377	0.507	0.043
7/77	260	0.349	0.029	1/81	487	0.655	0.055
8/77	274	0.368	0.031	2/81	628	0.935	0.072
9/77	212	0.294	0.024	3/81	500	0.672	0.057
10/77	280	0.376	0.032	4/81	537	0.746	0.061
11/77	340	0.472	0.039	5/81	386	0.519	0.044
12/77	368	0.495	0.042	6/81	321	0.446	0.037
1/78	718	0.965	0.082	7/81	492	0.661	0.056
2/78	480	0.714	0.055	8/81	656	0.882	0.075
3/78	420	0.565	0.048	9/81	73	0.101	0.008
4/78	410	0.569	0.047	10/81	570	0.766	0.065
5/78	290	0.390	0.033	11/8	416	0.578	0.047
6/78	290	0.403	0.033	12/8	352	0.473	0.040
7/78	360	0.484	0.041	1/82	830	1.116	0.095
8/78	360	0.484	0.041				

TABLE XVI Project A Curves



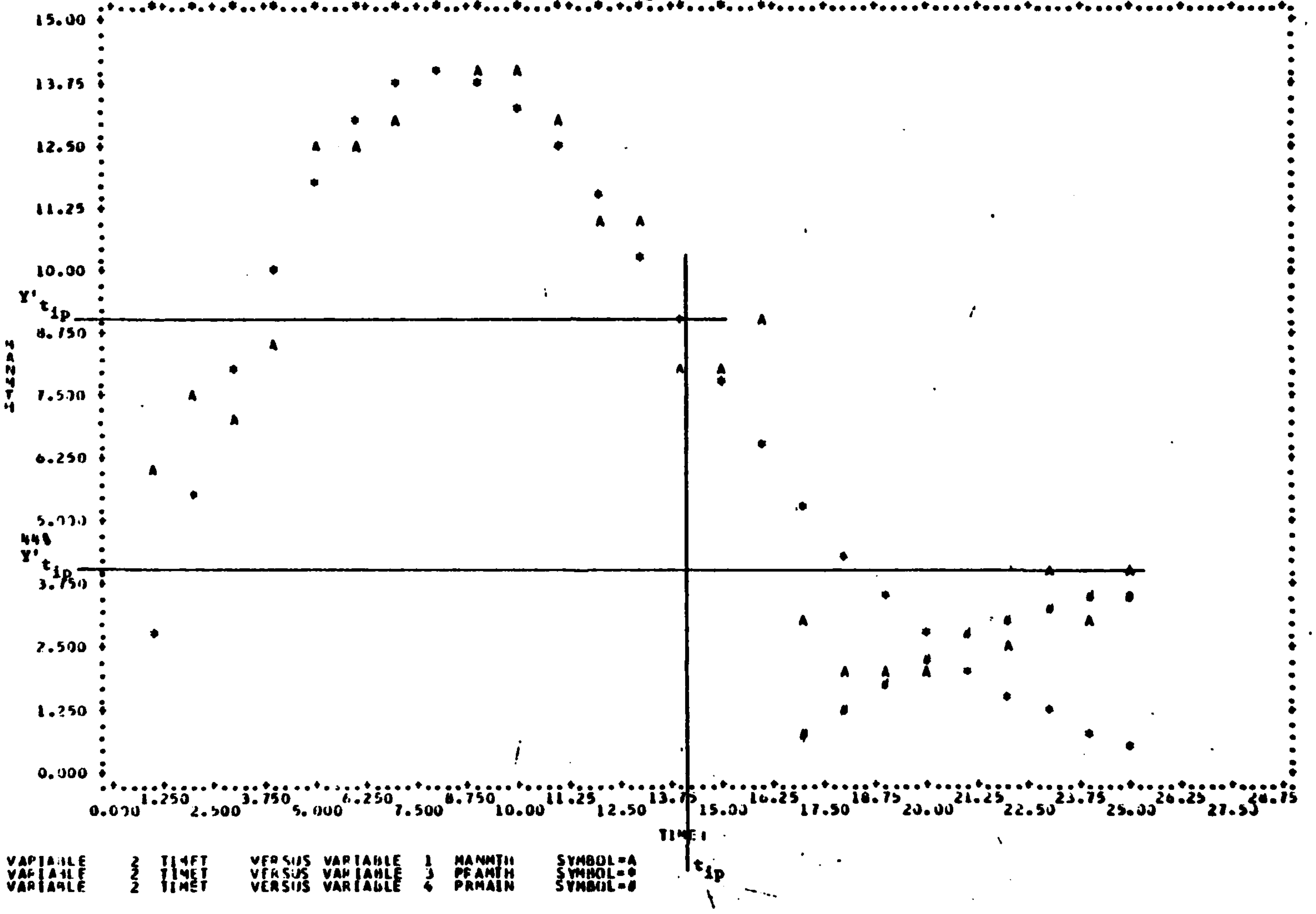
```
VARIABLE   2  TIMET   VERSUS VARIABLE  1  MANMTH   SYMBOL=A
VARIABLE   2  TIMET   VERSUS VARIABLE  3  PRAMTH   SYMBOL=*
VARIABLE   2  TIMET   VERSUS VARIABLE  4  PRMAIN   SYMBOL=#
```

TABLE XVII   Project B Curves



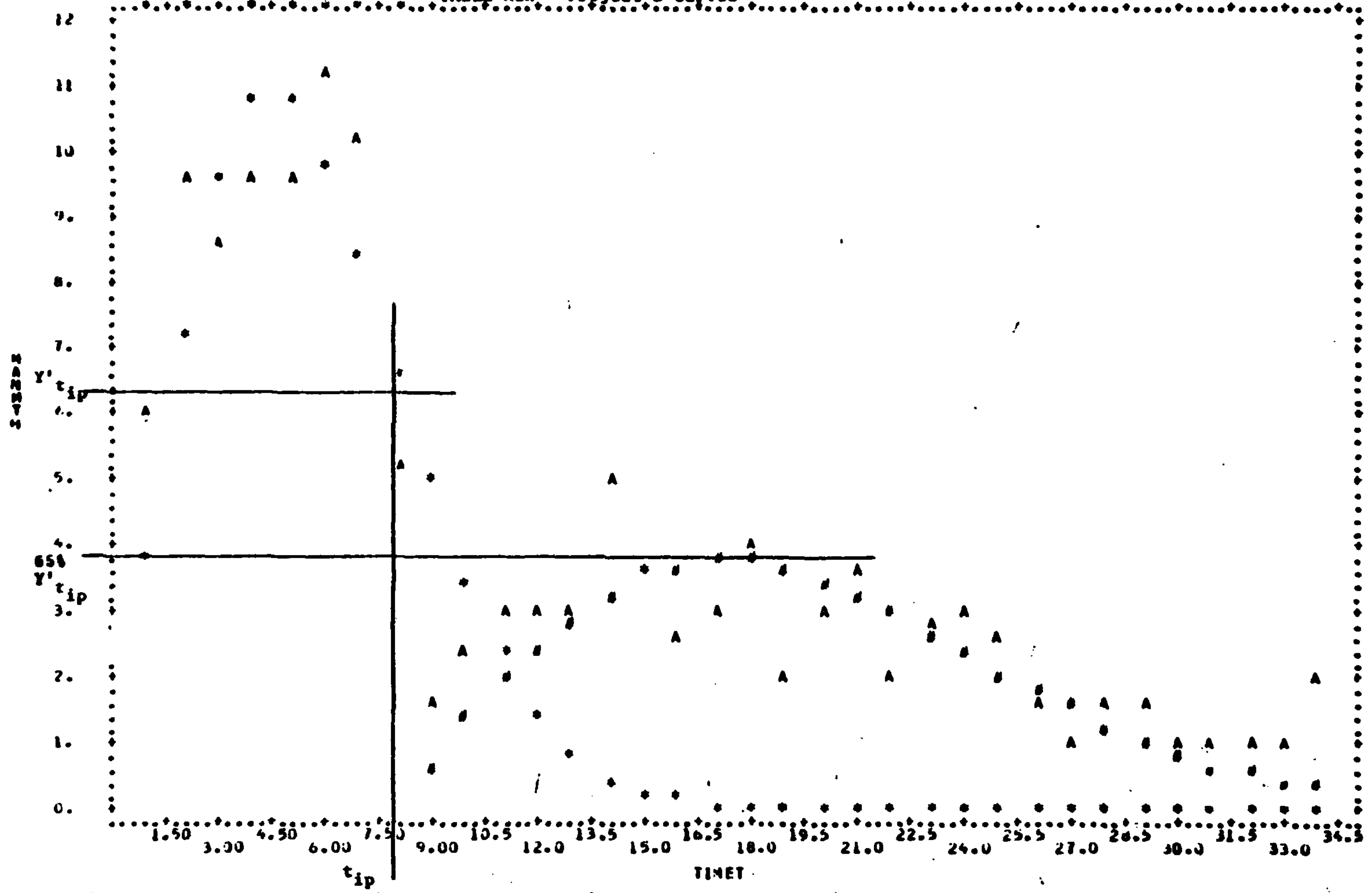
```
VARIABLE   2  TIMET   VERSUS VARIABLE  1  MANMTH   SYMBOL=A
VARIABLE   2  TIMET   VERSUS VARIABLE  3  PRAMTH   SYMBOL=*
VARIABLE   2  TIMET   VERSUS VARIABLE  4  PRMAIN   SYMBOL=#
```

TABLE XVIII Project C Curves



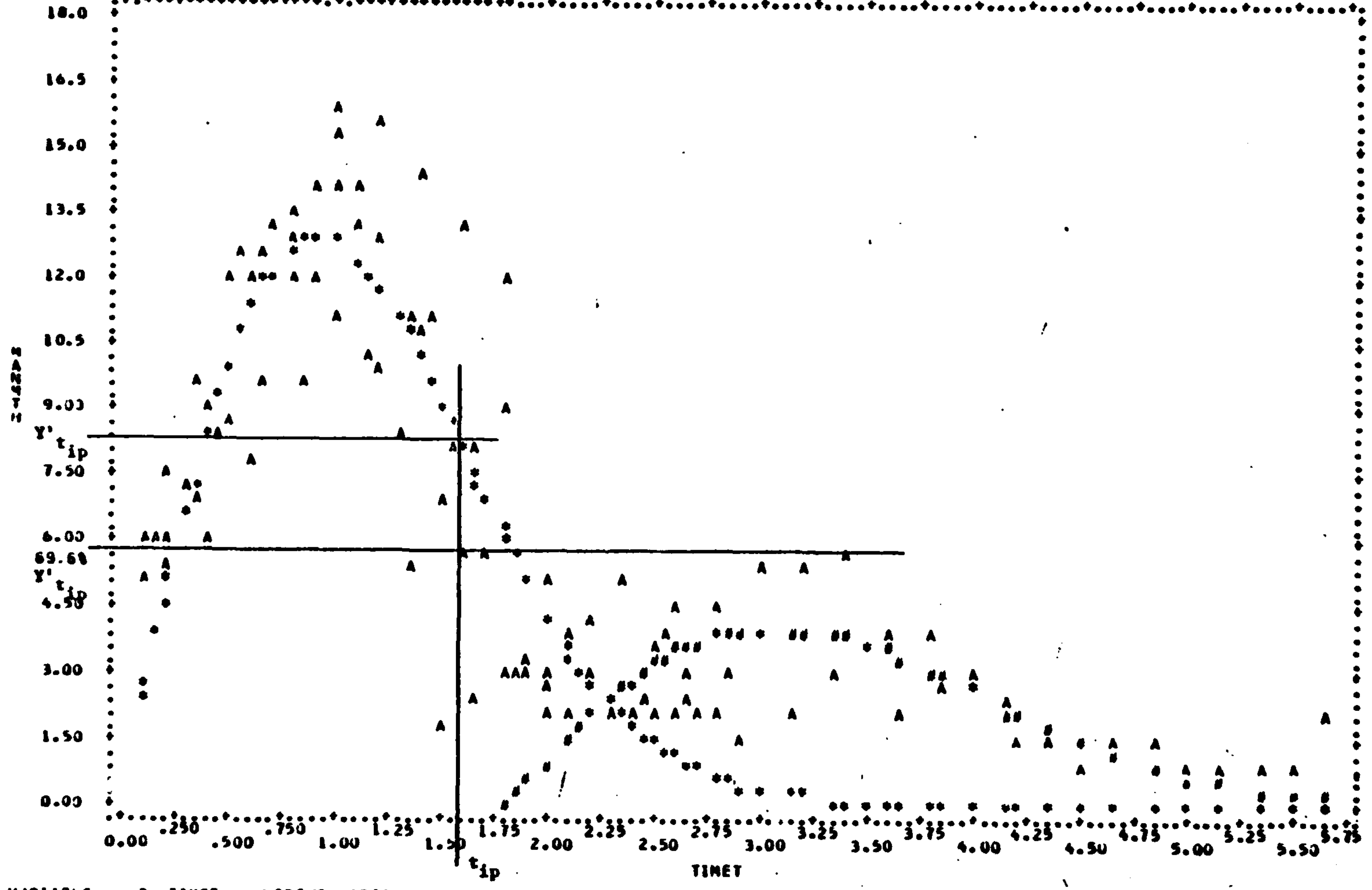
```
VARIABLE   2  TIMET    VERSUS VARIABLE  1  MANMTH    SYMBOL=A
VARIABLE   2  TIMET    VERSUS VARIABLE  3  PFAMTH    SYMBOL=*
VARIABLE   2  TIMET    VERSUS VARIABLE  4  PRMAIN    SYMBOL=#
```

TABLE XIX Project D Curves



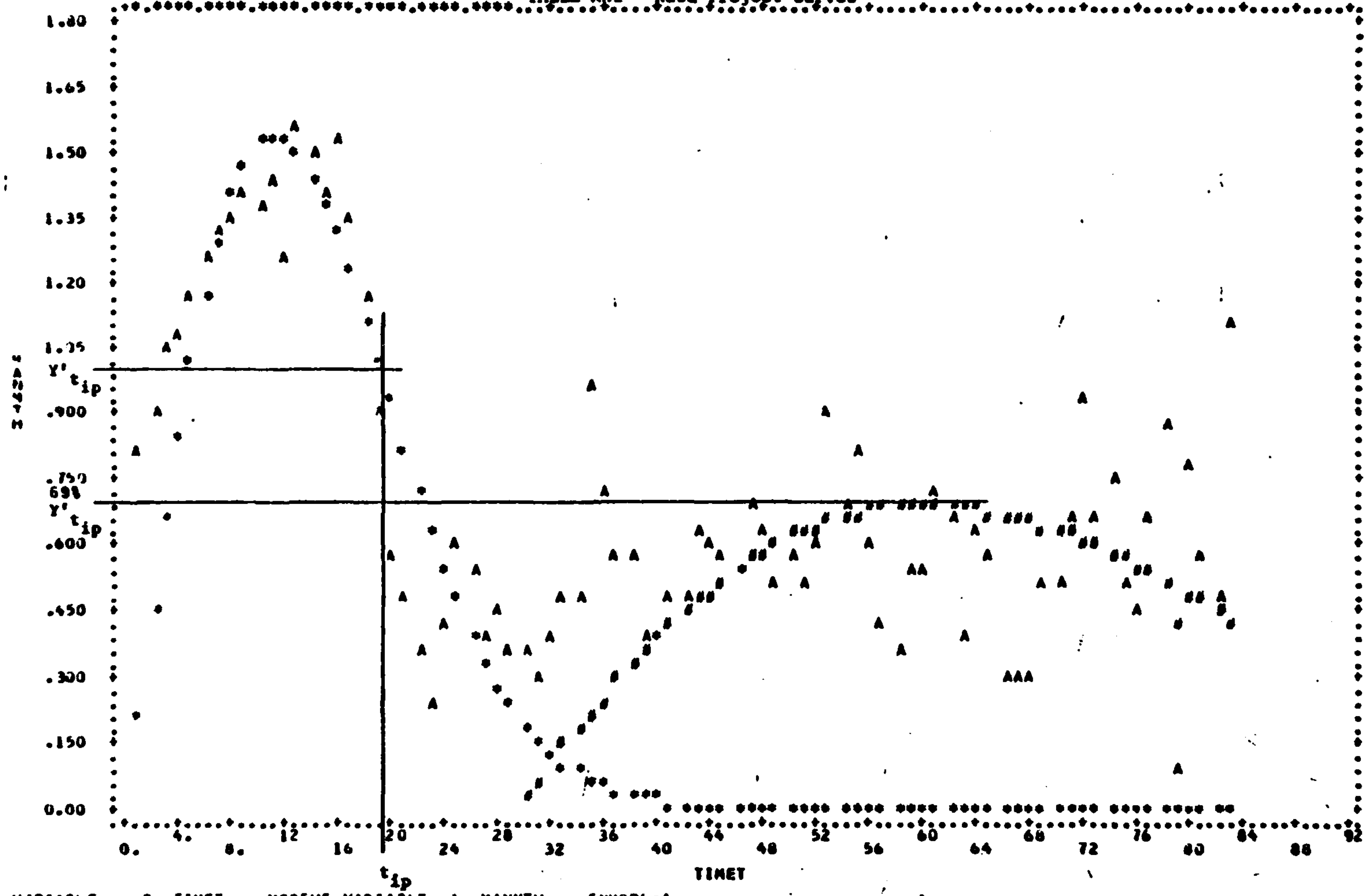
```
VARIABLE    2  TIMET    VERSUS VARIABLE  1  MANMTH    SYMBOL=A
VARIABLE    2  TIMET    VERSUS VARIABLE  3  PRAMTH    SYMBOL=*
VARIABLE    2  TIMET    VERSUS VARIABLE  4  PRMAIN    SYMBOL=#
```

TABLE XX   Combined Project A-D Curves



```
VARIABLE    2  TIMET    VERSUS VARIABLE  1  MANMTH    SYMBOL=A
VARIABLE    2  TIMET    VERSUS VARIABLE  3  PRAMTH    SYMBOL=*
VARIABLE    2  TIMET    VERSUS VARIABLE  4  PRMAIN    SYMBOL=#
```

TABLE XXI Nasa Project Curves



```
VARIABLE    2  TIMET     VERSUS VARIABLE  1  MANMTH     SYMBOL=A
VARIABLE    2  TIMET     VERSUS VARIABLE  3  PRAMTH     SYMBOL=*
VARIABLE    2  TIMET     VERSUS VARIABLE  4  PRMAIN     SYMBOL=#
```

LIST OF REFERENCES

1. Freeman, Peter, and Wasserman, Anthony I. Tutorial on Software Design Techniques Third Edition, IEEE Catalog No. EHO 161-0, 1980.

2. Ibid., pp. 5-16.

3. Wegner, P., Research Directions in Software Technology, Third International Conference on Software Engineering, Proc., pp. 243-259, 10-12 May 1978.

4. Boehm, B., "Software and its Impact: A Quantitative Assessment", Datamation, V.19, no.5, pp.48-59, May 1973.

5. Waina, R.B., Foreman, G.L., Bangs, A.P., and Green, J.L., "Predictive Scftware Cost Model Study", Air Force Systems Command, pp. 18-20, 11 June, 1980.

6. Glass, Robert L., and Noiseux, Ronald A., Software Maintenance Guidebook, Prentice Hall, 1981.

7. Green, James F. and Selby, Brenda F., Dynamic Planning and Control of Software Maintenance: A Fiscal Approach, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1981.

8. "Federal Agencies' Maintenance of Computer Programs: Expensive and Undermanaged", General Accounting Office, AFMD-81-25, February 26,1981.

9. Ibid., pp. 28-29.

10. Putnam, Lawrence H., Tutorial-Software Cost Estimating and Life Cycle Control: Getting the Software Numbers, IEEE Computer Society, 1980.

11. Putnam, Lawrence H. and Wolverton, Ray W., Tutorial-Quantitative Management Software Cost Estimating, IEEE Computer Society, November 1977.

12. Putnam, Lawrence H., Op.Cit.

13. Ibid., pp. 313-314.

14. Ibid., pp. 316-317.

15. Ibid., pp. 71-90.

16. Parr, F.N., "An Alternative to the Rayleigh Curve Model for Software Development Effort", IEEE Transactions on Software Engineering, Vol. SE-6, no.3, May 1980.

17. Department of the Army DA Pamphlet No. 18-8, A Software Resource Macroestimating Procedure, February 1977.

18. Belady, L.A. and Lehman, M.M., "A Model of Large Program Development", IBM Systems Journal, V. 15, pp. 225-252, 1976.

19. Ibid., pp. 225-252.

20. Ibid., pp. 225-252.

21. Jensen, Randall W., "A Macro-level Software Development Cost Estimation Methodology", Fourteenth Asilomar Conference on Circuits, Systems, and Computers, 17-19 November 1980.

22. Ibid., pp. 320-325.

23. Ibid., pp. 320-325.

24. Ibid., pp. 320-325.

25. Ibid., pp. 320-325.

26. Thibodeaux, Robert, "An Evaluation of Software Cost Estimating Models", Rome Air Development Center, Griffis AFB, NY, June 1981.

27. Wolverton, R.W., "Airborne Systems Software Aquisition Engineering Guidebook for Software Cost Analysis and Estimating", Wright-Patterson AFB, Ohio, September, 1980.

28. Putnam, Lawrence H., Tutorial-Software Cost Estimating and Life Cycle Control: Getting the Software Numbers, IEEE Computer Society, 1980.

29. Ibid., pp. 76-77.

30. Ibid., pp. 76-77.

31. Ibid., pp. 11-97.

32. Norden, Peter V., "Usefull Tools for Project Management", Management of Production, M. K. Starr (editor), pp. 71-101, Penguin Books, Inc., 1970.

33. Green, James F. and Selby, Brenda F., Op.Cit.

34. Black, Rachael K.E., Curnow, Richard P., Katz, Robert, Gray, Malcolm, and Gray, D., "BCS Software Production Data", Rome Air Development Center (ISIM), Griffis AFB, NY, March 1977.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Defense Logistics Studies Information Exchange U. S. Army Logistics Management Center Fort Lee, Virginia 23801	2
3.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
4.	Department Chairman, Code 54 Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
5.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
6.	LCDR Ron Modes, Code 52MF Department of Computer Science Naval Postgraduate School Monterey, California 93940	5
7.	Dr. Dan C. Boger, Code 54BK Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	5
8.	Dr. Norman F. Schneidewind, Code 54SS Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	1
9.	Dr. Robert Grafton, Code 433 Office of Naval Research 800 N. Quincy St. Arlington, Virginia 22217	1
10.	Commander Naval Research Laboratory Attn: Code 7590 Washington, D.C. 20375	1
11.	Dr. Victor R. Basili Computer Science Department University of Maryland College Park, Maryland 20742	1
12.	Mr. Lawrence H. Putnam Quantitative Software Management, Inc. 1057 Waverly Way McLean, Virginia 22101	1

13. Dr. Willa K. Weiner-Ehrlich 1
c/o Vincent Rupolo
1 Bankers Trust Plaza
32nd Floor
New York, New York 10006

14. LCDR Stephen L. Tody, USN 3
Chief of Naval Operations (OP11-OP112-C2)
Navy Department
Washington, D.C. 20350

15. Captain Ray A. Hodgson, USA 3
La Cygne, Kansas 66040

16. 1st LT. Joseph N. Reinhart III, USMC 1
Marine Corps Development and Education Command
Quantico, Virginia 22134

17. Lieutenant Richardo Arana C. 1
Ministerio de Marina
Central Procesamiento de Datos de la Marina
de Guerra
AV. Salaverry S/N
Lima, Peru
South America

18. Ms. Cheryl Maloney 1
STARFIARS
United States Army Computer Systems Command
Ft. Belvoir, Virginia 22060

19. Mr. Robert Jones 1
STARFIARS
United States Army Computer Systems Command
Ft. Belvoir, Virginia 22060